FROM SHAPE TO FUNCTION: ACQUISITION OF TELEOLOGICAL MODELS FROM DESIGN DRAWINGS BY COMPOSITIONAL ANALOGY

A Dissertation
Presented to
The Academic Faculty

by

Patrick W. Yaner

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

College of Computing
Georgia Institute of Technology
December, 2007

FROM SHAPE TO FUNCTION:
ACQUISITION OF TELEOLOGICAL MODELS FROM DESIGN DRAWINGS BY COMPOSITIONAL ANALOGY

Approved by:

Ashok K. Goel, Advisor
College of Computing
Georgia Institute of Technology

Ronald Ferguson
College of Computing
Georgia Institute of Technology

Ashwin Ram
College of Computing
Georgia Institute of Technology

Charles Eastman
College of Architecture
Georgia Institute of Technology

Nancy Nersessian
College of Computing
Georgia Institute of Technology

Janice Glasgow
School of Computing
Queen's University

Date Approved: September 28, 2007

To my parents,

Bill and Maureen,

for their unwavering love and support

ACKNOWLEDGEMENTS

Although it is my name that goes on the front, there are many people who deserve credit at the completion of an undertaking as large and complex as a doctoral dissertation, and inevitably many who deserve credit may escape unacknowledged, but I shall do my best to give all that is due.

It goes almost without saying that any doctoral student owes a great intellectual debt to his advisor, and I do to mine, Professor Ashok Goel, whose fingerprints can be seen all over this document and all over the work, not simply in my drawing on previous work in SBF but even in the very idea of using analogy to construct shape representations at multiple levels of abstraction. A large debt is also owed to my committee members, to Dr. Ron Ferguson for many fruitful discussions at many stages of the work about shapes, shape representation, analogy, and (perhaps especially) about Lisp programming strategies; to Professor Nancy Nersessian for much fruitful discussion about design and analogy and much else besides both in and out of the classroom; to Professor Charles Eastman for his helpful discussions and suggestions regarding both SBF as well as shape grammars and the shape representation more generally; to Professor Janice Glasgow for many helpful suggestions and comments that helped me to a much better formulation of my hypotheses; and to Professor Ashwin Ram for questions and comments that led me to a much better evaluation of the work; and of course to all for many comments and suggestions, related work, and valuable perspectives. There is no doubt that the work is much stronger than it might have been for this input. And, as all do say but is always a heartfelt sentiment at the end of such an undertaking as a doctoral dissertation, the mistakes that remain are mine alone.

Jim Davies, fellow (former) student and colleague was more than once a valuable sounding board for my sometimes wacky ideas about potential directions, and has always been ready with a good critique when it is needed. Another fellow student, Idris Hsi, was an enormous help in getting my dissertation proposal, and the oral presentation in particular, into something resembling good shape, as well as someone I came to occasionally for much-needed advice. Many other friends and colleagues have been a source of support besides, too many to name, but I would especially

like to thank the members of the INCITE Lab: Idris Hsi once again, Vernard Martin, Jeff Rick, Eli Tilevich, David Zook, and others. The dialogue and exchange of ideas amongst this group was a constant source of inspiration as well as wisdom that often encouraged me to stretch the boundaries of my knowledge and experience in innumerable ways.

When one looks back over the long sequence of small decisions that make up a career path, one sees in hindsight that certain individuals played roles that turned out to be instrumental in shaping one's own career path and even one's attitudes. It is scarely possible to name all those who have been of such key importance, but certain individuals always stand out. It was at North Carolina State University in Professor David Austin's philosophy of science class that I first realized just how deeply and intensely interested in science, philosophy, and cognition I really was, and it was in Professor James Lester's artificial intelligence class that I got my first exposure to the field that would eventually become my area of expertise that same semester, and of course it was from so many others besides, teachers and friends at NC State, that I learned so much and drew so much inspiration and ultimately started down the path that has led me to where I am today.

And finally, through it all, my parents, Bill and Maureen Yaner, were a constant source of strength and encouragement.

What little wisdom I have retained I owe to the many fruitful conversations, the inspiration, and the support and encouragement of all of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Visual media are of great importance to designers. Understanding a new design, for example, often means understanding a drawing. From the perspective of artificial intelligence, this implies that automated knowledge acquisition in computer-aided design can productively occur using drawings as a knowledge source. However, this requires machines that are able to interpret design drawings.

I view the task of interpreting drawings as one of constructing a teleological model of the design depicted in the drawings, where the model enables causal and functional inferences about the depicted design. I have developed a novel analogical method for constructing a teleological model of a mechanical device from an unlabelled 2D line drawing. The source case is organized in a Drawing Shape Structure Behavior Function (DSSBF) abstraction hierarchy. This knowledge organization enables the analogical mapping and transfer to occur at multiple levels of abstraction.

Given a target drawing and a relevant source case, my method of compositional analogy first constructs a graphical representation of the lines and the intersections in the target drawing, then uses the mappings at the level of line intersections to transfer the shape representations from the source case to the target. It next uses the mappings at the level of shapes to transfer the structural model of the device from the source to the target. Finally, the mappings from the source to the target structural model enable the transfer of behaviors and the functional specification from source to target, completing the analogy and yielding a complete DSSBF model of the input drawing. The Archytas system implements this method of compositional analogy and evaluates it in the domain of kinematic devices such as piston and crankshaft devices, door latches, and pulley systems.

CHAPTER I

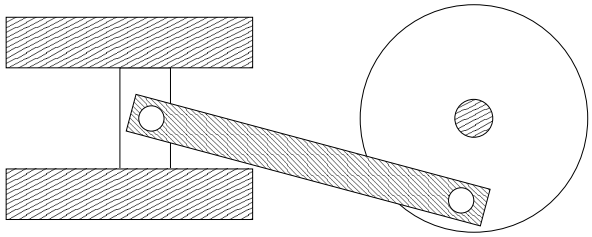
OVERVIEW

1.1 Motivation and Goals

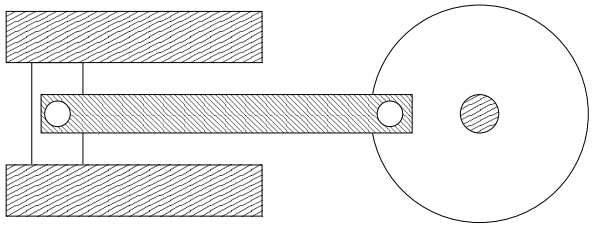
Visual media can be of great importance for designers, and so for example understanding a new design often means understanding a drawing. In artificial intelligence, this implies that knowledge acquisition in computer-aided design can productively occur using drawings as a knowledge source. However, for this to occur requires machines that are able to interpret drawings of designs.

In this work I view the task of interpreting drawings as one of constructing a model of what the drawing depicts; the model enables higher-level (i.e. non-visual) inferences regarding the depicted content of the drawing (see Glasgow et al. [44] for different perspectives on diagram understanding). For example, in the context of CAD environments, the input to the task may be an unannotated 2-D line drawing depicting a kinematics device, and the output might then be a teleological model of the device—i.e. a specification of the structure of the device (the configuration and properties of components) as well as the causal processes by which the device's function is achieved. Current methods for inferring a model from a drawing [3, 29] rely on domain-specific rules, and furthermore, only go so far as constructing a structural model from which causal inferences may be drawn. I propose first of all to infer a model by analogy to a similar drawing whose model is known, and furthermore for this model to be causal and teleological rather than merely structural.

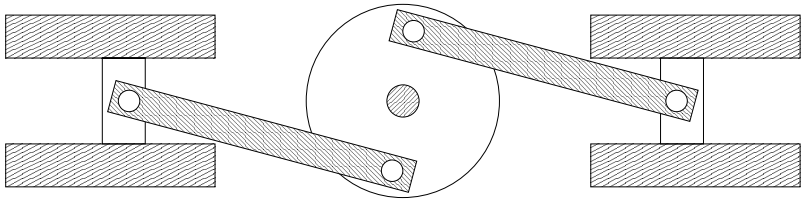
Let's consider the task of mapping the source drawing illustrated in figure 1(a) to the similar target drawing illustrated in figure 1(b). If we have a low-level geometric reasoner to recognize the geometric elements and spatial relations among them, then we can treat this representation as a labelled graph: A contains B, C is adjacent to D, and so on. A graph-theoretic method for analogy-based recognition may then be used to find a consistent mapping between the graphs representing the source and target drawings. However, the method runs into difficulty for the target drawings shown in figures 1(c) or 1(d) with figure 1(a) as the source drawing. In this problem, the *number* of components, and thus the number of shapes, is different, and either the graph-theoretic method



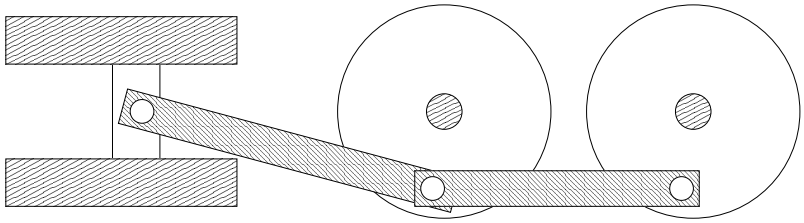
(a) A sample source drawing of a piston and crankshaft



(b) A target drawing of the same device with the piston now at the top of its range of motion



(c) A sample target drawing of a double piston and crankshaft



(d) A sample target drawing of a a device with two crankshafts operating off of a single piston

Figure 1: Piston and crankshaft example drawings

would have to relax the constraint of one-to-one mapping, or else the analogy would have to be performed twice in order to transfer a model successfully from figure 1(a) to figure 1(c) or 1(d). Either option requires reasoning at another level of abstraction in order to control the process: if one-to-many mappings are allowed, than how many of each item should a reasoner look for, and under what constraints? if the analogy is to be performed twice, then how does the reasoner know that twice is enough, and how are these multiple analogies brought together into one model?

To address the above difficulties, my method of compositional analogy performs analogy at multiple levels of abstraction. The method first gathers individual lines, circles, arcs, and intersection points into shapes, and then finds mappings between source shapes and the target drawing at this level of intersections. Then, it groups these mappings using symmetry and transfers whole shape patterns from the source to the target. This produces a mapping at the shape level, and using this it transfers the structural model as implied by the shape-level mapping. This gives a structure-level mapping, enabling a further transfer of causal process and teleology in similar fashion. Mapping and transfer thus work together *up* to each subsequent level of abstraction: mapping at one level enables transfer at another level, and this transfer defines a further mapping at the next level, and so on. Each model element transferred from source to target implies a mapping from the old model element to the new one in the target, setting up a transfer at the next level.

Given a drawing, then the knowledge to be inferred is both *visual* (that of shapes) and *teleological*. It is also important to note that the one must come before the other: until the visual knowledge is inferred the teleological knowledge cannot be inferred. However, teleology in a mechanical domain cannot simply be deduced from a 2D representation of shape alone. Additional information is required but not immediately available within the input alone. As such, I propose to perform the inference by analogy. The hypotheses then have to with both the *method* or process of analogical inference and the *knowledge* or content of the inferences. These hypotheses are:

- *Method Hypothesis*: in order to analogically infer the teleology of a device depicted in a drawing, a reasoner must perform analogical inference at several levels of abstraction, using the inferences at one level as a basis for those at the next level, from shape through structure, causality, and teleology

- *Knowledge Hypothesis*: in order to represent the teleology of a device depicted in a drawing, a reasoner must capture knowledge of the shapes present in the drawing, the structural elements they depict, the causal processes of these elements, and the teleology of the device

To this I will add the following hypotheses regarding the levels of abstraction and the stages of the reasoning process:

- *Dependency of Structural Knowledge on Shape*: Given a correct representation of the shapes in a target drawing, the correct structure may be inferred
- *Dependency of Causal Knowledge on Structural*: Given a correct representation of the structure of the device depicted in a drawing, a correct causal processes may be inferred
- *Dependency of Teleology on Causal Knowledge*: Given a correct representation of the causal processes of the device depicted in a drawing, a correct representation of the teleology may be inferred.

These hypotheses characterize the dependency of each level of abstraction on the level below it, and thus the need for analogy at several levels of abstraction, and in a reasoning process that starts at the lowest level and moves upward.

I address these hypotheses with my method of compositional analogy. The *Archytas* system implements this method. Although analogical reasoning in general involves the tasks of retrieval, mapping, transfer, evaluation, and storage, Archytas implements only the tasks of mapping and transfer. So, given a target drawing, I assume Archytas already knows of the source drawings relevant to the target. This work attempts to evaluate this theory of compositional analogy using experiments carried out with the Archytas visual analogical reasoning system.

1.2 From Drawing to Function By Analogical Transfer

The domain in which I plan to work is that of mechanical design. In general, knowledge of the design of a device may be classified into three basic kinds of knowledge:

1. *Structural* knowledge: what are the components of the device, how to they connect to and interact with one another, what are the relevant dimensions and parameters that describe the device?

2. *Behavioral* knowledge: how does the system behave? What are the causal processes and mechanisms? What motions produce what other motions?

3. *Functional* knowledge: what aspects of behavior are relevant to the usage of a device? What motions are those that the device is expected to produce, and how are these realized in the behaviors of the device?

These three kinds of knowledge of a device can be captured in a model known as a Structure-Behavior-Function (SBF) model [47, 12, 6]. These models will be discussed in greater detail in chapter 3. The structural model is a decomposition of the device into components, quantities, and structural relations between them, giving the important properties of of each component and quantity and the parameters that can vary in behavior. The behavioral model is a qualitative state-transition model, showing the major transitions between qualitative states of a particular component or quantity, and their immediate causes.[1] The functional model is a specification of those constraints on the behavior (and, by implication, structure) that are required for the operation of the device in question.

To this I add *drawing* and *shape*: a drawing depicts a device that can be represented using an SBF model. In particular, drawings show the components of a device and how they relate to each other. *Shapes* are those parts of a drawing that depict components and exhibit these relations. In particular, the models distinguish *basic* from *composite* shapes: basic shapes depict components, and composite shapes depict connections between components. If a connection between components can be treated as a relation, then it may be depicted by that composite shape made up of the basic shapes depicting the connected components (this will become more clear in subsequent chapters). Figure 2 shows this organizational scheme, from drawing on up through function, which I call *Drawing Shape Structure Behavior Function* (DSSBF) models.

1.2.1 Methods and Knowledge Organization

Archytas begins with a target drawing and a source design case. Each design consists of a teleological (SBF) model and a drawing that is associated with that model via the intermediate abstraction of shape.

[1] I.e., with respect to terminology, in SBF the *behavioral* model is a representation of the *causal* processes at work in a device that achieve its function.

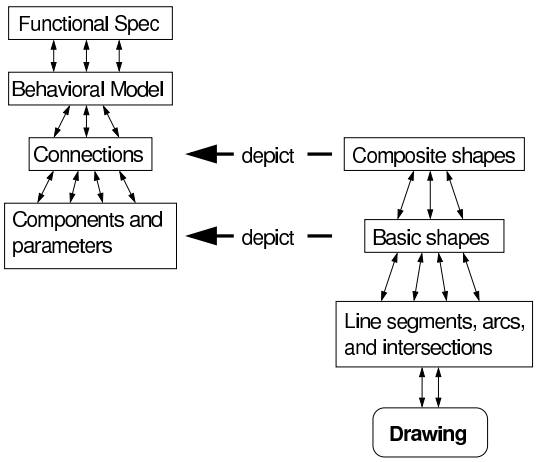


Figure 2: The multi-level hierarchy of drawing, shape, and teleology

In order to transfer and adapt the model to the target, Archytas must first infer the shapes present in the drawing, and from these, since shapes depict structural elements, infer which structural elements are being depicted in the drawing. This is the *first* step: from drawings to structure. Once the structure has been inferred, Archytas must infer the behaviors of these structural elements and, on the basis of these behaviors, the device's function. This is the *second* step: from structure to function.

These two steps are determined by the nature of the knowledge to be transferred: both visual and teleological. The inferences themselves define the *task*, which then leaves two questions for each of the two steps:

1. *Method*: By what procedure or method might the desired inferences be drawn?

2. *Knowledge*: How might one organize the knowledge so as to enable the system to draw these inferences?

If the two steps we call, respectively, step A (from drawing to structure) and step B (from structure to function), then questions 1A and 2A raise two problems with regard to step A, and questions 1B and 2B raise two problems with regard to step B. These are the problems that this research will consider.

1.2.2 Research Problems

1.2.2.1 From Drawing to Structure by Analogical Transfer

The main problem for step A is: given a drawing of a mechanical device, how might a reasoner construct a model of the structure of the depicted device, that is, a representation of the components, their properties, and their interconnections? As pointed out above, current methods rely on domain-specific rules, using either production systems or Bayesian methods, or some similar approach. In general, they treat the input as being written in a kind of two-dimensional *language*, and so the problem is then to represent the symbols in this language and the rules of composition. One approach [29] treats this as a problem of logic and representation, another [3] as a problem of learning and induction. But both assume that such a language exists. In diagrammatic domains such as electrical circuit diagrams or UML, often such a language does exist, but in general such languages might not exist.

My approach is to forego visual symbols or language constructs, and instead ask what kinds of visual patterns have been used in previous drawings to depict what kinds of elements. This does not actually require assembling a vocabulary of shapes or visual symbols, only an association between elements of *particular* drawings and *particular* models of the devices they depict. Given a set of cases, it may be possible in principle to reconstruct the visual language that would result in the same computation, and in that sense one might suggest that even this case-based method defines a language of a sort. However, this is an empty observation: even if a language can always be shown to exist, the question remains as to how this language ought to be defined, and the advantage of a case-based method is that it only requires the furnishing of examples; it does not require a search for a priori rules to which the examples all happen to conform. The difference thus is one of underlying knowledge assumptions: do we have available to us a set of rules to which the input drawings conform, or do we have available to us examples of input drawings that have been parsed into their elements, but not necessarily any rules to which they happen to conform? This is the basic question (suitably generalized) behind any case-based or analogical method.

There is a limitation to this approach which must be pointed out at the start. Some complex patterns may in fact be better represented by model-based approaches rather than case-based. For

instance, electrical circuit diagrams can connect elements with paths of arbitrary complexity, something which would confound any analogical method that uses structural alignment. However, drawings of a depictive nature, such as one sees in mechanical design, are not always following a visual language or rule set per se, even if they are following certain conventions, and so the analogical approach is capable of working with more success.

It is important to note that this approach does not require a vocabulary of basic geometric shapes, such as rectangles and rhombuses, but only a scheme for representing how such geometric shapes are arranged. A particular model element—say, the piston in the piston and crankshaft device considered above—may be represented by a rectangle, or perhaps by a more (or less) complex shape. It is important to be able to find it in a target drawing in any case, and as such this work treats the recognition and representation of shapes and spatial relations as a problem for analogical reasoning.

Applying the two questions from above, we get the following:

1. By what *method* might a reasoner infer a structural model of the device depicted in a given input drawing?

    - From an input drawing, the agent can first attempt to analogically map and transfer individual shapes that depict structural elements in a similar, known drawing; this enables a shape-level mapping from which the structural elements themselves can be transferred.

2. How might a reasoner organize the *knowledge* of a source case, a known drawing and the model of the depicted device, so as to enable the analogical transfer of shapes and then structure?

    - Shapes can serve as an intermediate abstraction between drawing and structure in the model. First, the individual lines and circles and circular arcs in the drawing can be aggregated into *basic shapes* that depict components, and which therefore are linked to the components in the model. Second, basic shapes can be further aggregated into *composite shapes* that depict connections between components, treating these as aggregations of components in the model, and these shapes linked to the connections they depict.

It is these two proposals together which define the approach of mapping and transfer iteratively up through several levels of abstraction. This theme recurs in step B, described next.

1.2.2.2 From Structure to Function by Analogical Transfer

The standard methods for inferring behavior from structure all orbit the notion of qualitative physical reasoning. [19, 30, 57]. The idea is to begin with a representation of the structure of a device and make causal inferences on this basis. That is, to attempt to answer questions that ask what the outcomes would be of certain events in some situation. This is generally done by deriving a *qualitative envisionment*, whereby a qualitative representation of the device's behavior is inferred and from which specific questions might be answered.

These methods cannot derive function, properly speaking, since function does not simply follow from the physics of a device but depends on how a device is intended to be employed. For instance, a piston engine turns the crankshaft, and as an engine (say in an automobile) that is its function, but it also generates heat, which in the simplest case at least is not its function but an effect of its normal operation. Both are causal processes and can be inferred by a sufficiently complete and well organized causal model, but how do we know that one is the function and the other is not?

Structure Behavior Function models [47] directly represent function as well as the causes of behaviors. They do this at a tradeoff: qualitative physical models can infer *different* behaviors using *different* initial conditions, but SBF models are limited to those causes and effects explicitly represented in the models. Finally, the method by which SBF models are (and have been in the past) inferred is analogical as opposed to deductive: a model of a new design is inferred by analogy to a similar model. This is both a strength and a weakness. It is a strength because there is as yet no well-known analogical theories of qualitative envisionment (but see PHINEAS [24]), and yet it is a weakness because it limits the kinds of inferences that are possible with SBF models.

Taking this discussion in hand, and applying the two questions from above, we then get the following:

1. By what *method* might a reasoner infer the function of a device whose structure is known, and is known by analogy to an existing device with a known model?

    - From the structural model and a mapping from this to another model, a reasoner might

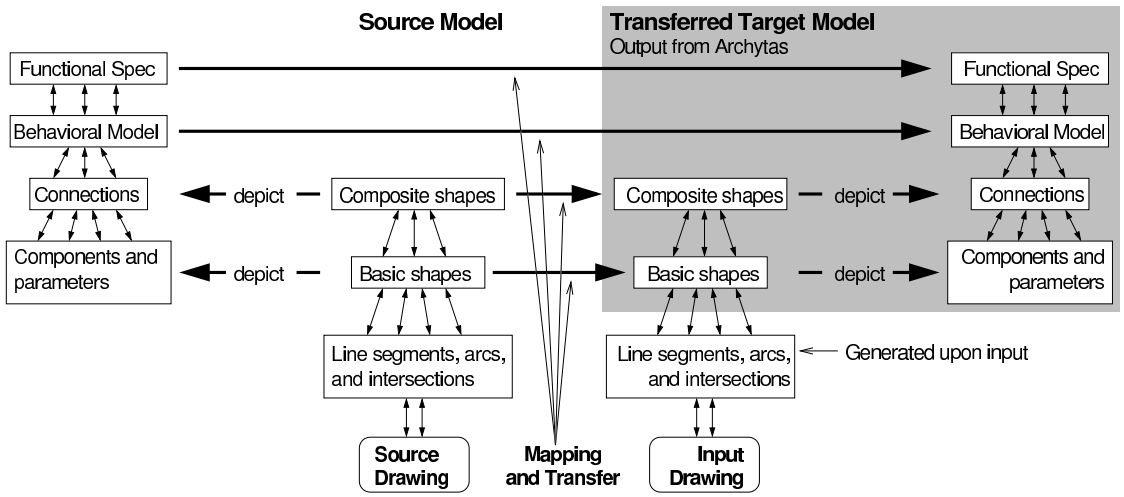


Figure 3: The compositional analogical process in the multi-level hierarchy

analogically transfer the behaviors of each component and, from this, abstract the function by analogy to the function of the known model.

2. How might a reasoner organize the *knowledge* of a source case so as to enable the analogical transfer of behavior and function?

   - Each mapped component can have a behavior in the source model and the function is then specified in terms of behavioral states of the components.

These two proposals together along with the two above (from drawing to structure) that define the approach of mapping and transfer iteratively up through several levels of abstraction: first the level of drawing, then that of shape, then that of structure, then behavior, and finally function. This approach is illustrated in figure 3.

1.3 Compositional Analogy

As according the above discussion, Archytas is divided into two stages: a *shape analogy* stage [87] and a *functional analogy* stage. The shape analogy stage begins with a drawing and a complete source DSSBF model including, most importantly, the source drawing and shape model. It first passes the drawing through a low-level geometric reasoner that generates a representation of the individual line segments, circles, and circular arc segments in the drawing. This is the *augmented line intersection graph*, described in detail in chapter 2. The purpose of this graph is to provide for

a representation of shapes as patterns over this graph, allowing structural alignment to be the basis for recognizing shapes. The shapes in the source, both composite and basic shapes, are treated as subgraphs of this graph, and so the next part is to compute the shape analogy by matching each shape individually to the intersection graph using backtracking constraint satisfaction. This constraint satisfaction algorithm is based on that of the *Geminus* system [86], with one major change: in Geminus, the target is treated as a pattern to be matched against the source, but in Archytas it's just the other way around: the source shapes are the patterns being matched against the target. This is discussed in more detail in chapter 4.

Once target shapes have been inferred, each one is mapped to the source from which it came, and the result is a shape-level mapping. This is the shape analogy that is the output of the first stage. The second stage then takes over and passes through three sub-stages: transfer of structure, transfer of behavior, and transfer of function. Each mapped shape implies a structural element to be transferred, and then mapped. Each mapped structural element may imply a behavior to be transferred, and then mapped, including all its states and transitions. The function of a device refers to particular behavioral states, and these, being mapped, then imply the device's function. The result of this is a mapping from the source to the target function, which in turn contains behavioral and structural mappings: a functional analogy.

From the above discussion then, the outline of the steps of the compositional analogy method implemented by Archytas are:

1. **Shape Analogy**: infers a mapping between source shapes and transferred target shapes that may not be one-to-one.

    (a) **Augmented Line Intersection Graph**: runs the target drawing through a low-level geometric reasoner that generates a representation of the line segments, circles, and circular arc segments in a drawing, computing all intersection points. This structure represents the most basic topological information in the drawing and allows a subgraph isomorphism test to determine presence or absence of a given shape

    (b) **Shape Mapping**: for each composite shape in the source, attempt to map it to the target, computing all mappings and grouping by symmetry; break each mapping into the basic

shape mappings of which it's composed and group these as well; transfer each shape once for each mapping group, and return the shape-level mapping as the shape analogy

2. **Functional Analogy**: infers mappings between the source structural, behavioral, and functional models and the transferred target models, again that may not be one-to-one.

   (a) **Structural Transfer**: for each mapped basic shape, transfer the implied structural component; for each mapped composite shape, connect the already mapped structural components; transfer non-depicted components and map them; return the mapping as a structural analogy

   (b) **Behavioral Transfer**: for each component or quantity that enters into a behavior, transfer that behavior, iterating through each state and transition, setting up mappings after each transfer; return the complete mapping as a behavioral analogy

   (c) **Functional Transfer**: for each of the behavioral states referred to in the source function, find the target states, and using these, construct a new target function, mapping the source to the target function. Return this as the functional analogy.

The input here is a target drawing and a source case as a DSSBF model, and the output is a target DSSBF model and a detailed mapping between the two models.

In this analogical reasoning process there are two kinds of composition going on. In the first place, the basic shape mappings are aggregated into composite shape mappings just as basic shapes are aggregated into composite shapes. And in the second place, mappings at a lower level imply transfer and thus mappings at a higher level. Thus, there is (1) composition by aggregation and (2) composition by abstraction. In both cases there are many different mappings and transfers going on at all levels. The input is a target drawing and a source drawing with a model already associated with it, and the output is a functional analogy containing, in turn, the behavioral, structural, and shape analogies that ultimately represent the acquired model of the input drawing.

1.4 Outline

The rest of the dissertation is organized as follows. Chapter 2 discusses the representation of lines and intersections in drawings and shape patterns, chapter 3 discusses the organization of the DSSBF

models, including a detailed review of the organization of SBF models, and runs through some detailed examples in this representation. Chapters 4 and 5 detail the algorithms that operate on these representations, respectively covering the first and second steps described above. Chapter 6 describes experiments and chapter 7 discusses some related work and chapter 8is the closing discussion and conclusion. That is:

1. **Overview**: this chapter
2. **Representation of Shapes**: describes and motivates the representation of basic and composite shapes using augmented line intersection graphs
3. **Organization of the Device Models**: reviews previous work on SBF and describes the organization of DSSBF models, including examples
4. **From Drawing to Structure**: the algorithms that infer the structural model from the input drawing by analogy
5. **From Structure to Function**: the algorithms that infer the functional model from the structural model by analogy
6. **Experiments**: discussion of the experiments run in the investigation of the above problems and hypotheses
7. **Related Work**: discussion of related literature
8. **Conclusion**: summary and conclusions

These chapters together provide a theory of drawing understanding in design by compositional analogy.

CHAPTER II

REPRESENTATION OF SHAPES

Since the task is the analogical comparison of drawings, and drawings contain shapes that depict model elements (components of the device, etc.), these shapes must be represented in such a way as to facilitate this comparison. When a drawing is represented symbolically for analogical comparison, symbols are associated with shapes. In doing so, the shape—which is in truth a composite geometric structure, a point set in the real plane—becomes an atomic entity. If analogy involves the alignment of symbol structures, then analogical comparison between two drawings can only be successful when these symbol structures are isomorphic. This opens up the way to a variation of the symbol mismatch problem, whereby two seemingly similar entities are represented by non-isomorphic symbolic structures and are thus incapable of analogical comparison. Put differently, what may interfere with the process of analogical comparison between drawings are (1) the *symbol mismatch* problem, tokens of different types within some shape type system that should be seen to be similar but cannot, or else (2) different levels of aggregation and abstraction between two representations, e.g. three stacked rectangles may be regarded as one complex shape, three simpler ones, or a set of intersecting line segments—one representation in the one drawing and another representation in the other drawing inhibits analogical comparison. The root of this problem is the following: the agent cannot compose line segments into shapes unless it already knows what shapes are supposed to be there.

The key here is to observe that the breaking up of a line drawing into shapes is already a form of interpretation: the agent is beginning to count discrete elements in the drawing and determine their relationships to each other. In other words, the representation of the shapes in a drawing is a function of the comparisons being made with that drawing. As such, this interpretation might also be done by analogy. Here I explore an analogical method for accomplishing this task. In particular, I explore a means of breaking up the atomicity of shapes labelled by symbols, and develop a method

of aggregating and abstracting the geometric structure of shapes in drawings, thus effecting a comparison between two geometric structures—shapes in drawings—at whatever level of aggregation and abstraction is most appropriate for the match. The discussion is centered around a mathematical notation that will be used in developing the representation proper.

To focus the discussion on geometric issues, here I depart momentarily from the concrete examples of design drawings and employ simple but abstract shapes. At the end of the chapter I will relate shapes to structure in SBF, and then the discussion will return to concrete examples.

2.1 Visual Versus Structural Similarity

Two drawings that are visually very similar or even isomorphic in some sense may be very different in terms of the composition of the knowledge structures that represent them in a computer. For instance, what looks like a rectangle made up of four line segments in one drawing may in fact be six or eight connected line segments, some of which are collinear and some of which are perpendicular and may not have been collected together into one shape in the internal representation. However, an analogy between these drawings, if it is expected to map a rectangle in one drawing to a rectangular shape in the other, must have some element or elements to compare with each other. In order for an agent to make an analogical connection between two drawings, and see that they are drawings of the same thing (or even merely similar things), that agent must be able to represent the visual structure of the drawings, i.e. the agent must be able to decompose the drawings into shapes, sub-shapes, composite shapes, and arrangements thereof. In short, visual isomorphism can only be decided in terms of an isomorphism of knowledge structures.

2.1.1 Previous Work

One approach to the problem of representing shape and structure, and in particular of deriving such a representation for an input drawing, is Ronald Ferguson's GeoRep system [29]. GeoRep is a reasoner for diagrammatic domains such as electrical circuit diagrams. It is organized as a two-stage forward chaining production system. It first reads in a line drawing (from, say, XFig[1]) and then first passes it through a "low-level relational describer" (LLRD), which attempts to reconstruct polygons

[1] See `http://www.xfig.org/`

and represent relations such as connectivity, parallelness, and perpendicularity of individual lines. These relations are then fed into a rule-based truth maintenance system, the "high-level relational describer" (HLRD), that employs a domain-specific ontology to construct a representation of the diagram's content in terms of specific rules regarding the composition of shapes in that domain. For instance, in electrical circuits, *and* gates and *or* gates have particular shapes that can be well represented by rules.

GeoRep represents a traditional knowledge-based approach to shape representation in drawings: devise an ontology and employ a reasoning engine of some sort to apply this ontology to particular drawings. The two stage nature is in recognition of the fact that certain relations are going to be basic and common to all domains, and therefore tedious to represent in rule sets. There are, however, alternative approaches, and in particular one can imagine a more data-driven approach

Christine Alvarado has done work on multi-domain sketch recognition [3], and developed a system called SketchREAD. Like GeoRep, SketchREAD is intended to be applicable across domains, although the mode of input is freehand sketches rather than diagrams drawn with vector graphics. In SketchREAD, a first stage processes the strokes in the sketch to identify, for instance, possible straight lines and circles, and then a second stage applies known shape patterns to the drawing, constructing bayes nets for each shape pattern in the visual ontology to determine the likelihood that a pattern applies to a given set of strokes. Here, the bayes nets play a role similar to the domain rules in GeoRep.

2.1.2 Issues

In both of these cases above, Ferguson's GeoRep and SketchREAD, there has been an assumption that a domain is a fixed set of visual symbol patterns that can be applied either with rules or bayes nets, both representing model-based reasoning approaches. Note that this is *not* a problem with the ontologies but *rather* it is a problem with the mode of reasoning in both cases: strict bottom-up processing with no other top-down influence besides the imposition of visual domain rules. By contrast, in depictive domains such as design drawings there may be no clear visual vocabulary to apply, and the task is rather to determine some kind of structure that a drawing is depicting, which may not necessarily assign a particular symbol to each discrete pattern of strokes or lines.

However, there is a deeper problem that this implies. Both have assumed that the interpretation of a set of lines or stokes as a shape is determined by the input alone given a domain ontology. For instance, with electrical circuits, we can define a set of classes such as "and gate," "or gate," and so on, and then particular drawings or sketches potentially contain instances of all these classes that must be known in advance, the reasoning process being one of confirming or disconfirming matches between instances and classes.

From a psychological standpoint, however, this assumption appears to be unwarranted. The most basic task in recognizing shapes in some domain is that of comparison, either explicitly between two drawings or else implicitly to known prototypes or models. There is evidence, however, that the representation of shape features is not prior to this process of comparison but rather something that happens along with the comparison or even as a consequence of the comparison.

Doug Medin, Robert Goldstone, and Dedre Gentner [62] have shown that, when human subjects attempt to compare similar shapes, the representations of the constituents are determined in the context of the comparison, and not prior to it. For instance, subjects described a certain blob figure as having *three* "prongs" or "fingers" when compared to one alternative and *four* when compared to another. Another figure subjects described alternatively as a circle held by pincers when compared with one alternative and a circle overlapping a square when compared to another. In these and other experiments, Medin et al. were able to show that the representation of the features of each figure could not have been constructed prior to the process of comparison, but rather were constructed as a part of the comparison process itself.

In artificial intelligence, a similar idea has been advanced by Hofstadter and his colleagues. In particular, Gary McGraw's LetterSpirit [61, 52] illustrates how the representation of a target is constructed as part of the transfer process rather than prior to it. LetterSpirit constructs typefaces by analogy: it takes a stylized seed letter as input and outputs an entire typeface that has the same style (or "spirit"). The system understands "roles", such as the crossbar of an f or a t. It makes new fonts by determining some attributes such as "role traits" (e.g. crossbar suppressed), and transferring these attributes to the other letters to build an entire alphabet in a new style. Thus, two letters are "the same" (e.g. both As) if a sufficiently similar set of roles can be reconstructed for each one. Representation of shape is part of transfer rather than prior to it.

What is needed, then, is an analogical theory of shape, and in particular a method for determining the structure depicted in a drawing by analogically decomposing the drawing into shapes rather than applying domain rules or patterns in an a priori fashion. That is to say, instead of treating drawings by the application of known classes to potential instances of those classes, we treat existing known drawings as containing instances that can be regarded as generic and applied then as generic patterns to other instances. The distinction, here, is that the known patterns are determined by the drawings that have previously been successfully processed (or with which memory has been initially populated) rather than by a set of classes assumed to be canonical and complete.

Towards this end, I explore here first the subject of the mathematical and geometric structure of drawings, shapes, and arrangements of shapes, with a view towards building up a terminology suitable for devising a proper representation of shape; and second, I construct the shape representation in detail and show how it can be associated with the structural portion of an SBF model. In doing so, I will retain certain elements of the above work. For instance, GeoRep's division into a low-level and a high-level reasoner is based on an attempt to model certain aspects of human visual reasoning, and in like fashion Archytas employs a first stage that is non-analogical to represent certain important low-level spatial relationships, and it is only the second stage where analogical comparison begins. In that way, Archytas may be seen as employing an analogical reasoning process in place of GeoRep's rule-driven HLRD.

2.1.3 Analysis

It is worthwhile to imagine for a moment a non-analogical method for solving the problem that this chapter addresses: attempting to represent shapes in the absence of any a priori knowledge of what the correct set of shapes ought to be. In general, if an agent is to take a set of potentially intersecting line segments (and circular arcs, etc.) and determine which aggregations and groupings correspond to shapes and which do not, then there are only two possible choices:

- Attempt to select for particular shape patterns in a top-down fashion
- Represent all possible groupings and aggregations in a bottom-up fashion and let the reasoner choose the "correct" ones at the next stage

The first one we have specifically ruled out *ex hypothesi*: the patterns to be used in a top-down fashion are not known at this stage. The second is thus our only choice, but then this makes the analogical mapping at the next stage—the analogy of shapes for facilitating the transfer of a structural representation—a great deal harder. Instead of mapping actual shapes to actual shapes, it's mapping actual shapes to merely *potential* shapes, and *all* potential shapes at that. It should be clear that the search space for this will be intractable in general, and this would produce a combinatorial explosion very quickly.

This brings us back to the first choice. In what ways can a top-down influence be used? The work cited above (both GeoRep and SketchREAD) have the same answer: the patterns are known by the domain and so the top-down influence is done by comparing all known patterns within the domain to the input drawing. In our case-based method, by contrast, we will be comparing only the particular patterns seen in a particular drawing, and so the search space is already much smaller.

However, GeoRep in fact does not use top-down influence to aggregate lines into polygons, it attempts to search for connected line segments to aggregate into polygons in the LLRD in the absence of any hints from the HLRD. As such, it can detect certain kinds of patterns only if that is the assumed interpretation of the domain; for instance, it can detect two overlapping rectangles as such, or see them as a set of three connected rectilinear regions, but it cannot choose between them based on what it learns from the diagram as it interprets it. The methods I describe here suggest a way that the HLRD in a system such as GeoRep might "tell" the LLRD how to find polygons in a top-down fashion, notwithstanding the differences in representation, and so select between different line interpretations (two overlapping rectangles, three rectilinear regions) based on context in a drawing rather than simply the domain.

2.1.4 Shape Grammars

Before moving on it is worth contrasting the approach taken in this work with that of shape grammars, a well-publicized theory in the design literature pioneered by George Stiny and James Gips [75, 72, 73, 74]. Shape grammars consist of production rules, like those of context-free grammars in computational linguistics, which involve transformations from one shape pattern to another usually

more complex pattern. Unlike grammars in computational linguistics they are primarily a *generative* method rather than an analytical one to be used for recognition. Nevertheless, it is possible to imagine them being used for recognition by running the rules backwards. First, it is worth mentioning that the representations I discuss here, as well as the mapping algorithm discussed in chapter 4, could be adapted for use in shape grammars; many of the problems raised by employing shape grammars in recognition would be identical, and these methods could address them, however as most shape grammar material has been published in the design literature rather than the computer science literature, it does not address such concerns. Often the systems, even generative ones which must nevertheless match the left-hand side of a rule to the drawing, punt on the questions raised in this chapter. For instance, Mark Tapia's GEdit system [77] uses a grid-based representation for shape patterns, greatly simplifying representation and all but eliminating the problem of mapping.

The differences between shape grammars and my approach are not merely ones of detail, however; as said before, the representations here described could be adapted for use in shape grammars, so what is the difference between this analogical or case-based approach and shape grammars? In fact, it is exactly that: shape grammars employ production rules where the method being considered here uses cases and analogical transfer. There are advantages and disadvantages to each. Shape grammars are not limited to the two levels of basic and composite shapes that Archytas must employ, but can represent arbitrarily deep levels of composition, even for a fixed grammar. On the other hand, from a knowledge engineering perspective, a shape grammar system for recognition and interpretation must capture all the potential combinations of lines that might make up a given pattern, whereas a case-based method need only have at least one example drawing in the knowledge base that covers each likely combination. It is the difference between a model-based approach, on the one hand, in which the model is represented explicitly in rules, where exceptions and alternatives will be hard to capture, and a case-based approach, on the other hand, where exceptions and alternatives are straightforward but systematic and recursive patterns (such as fractal patterns that are well captured by shape grammars) may turn out to be much harder to cover by a set of appropriate examples. To a large extent the choice of domain may drive the decision, and in this case the domain has (thus far) proved easier to capture with examples than rules.

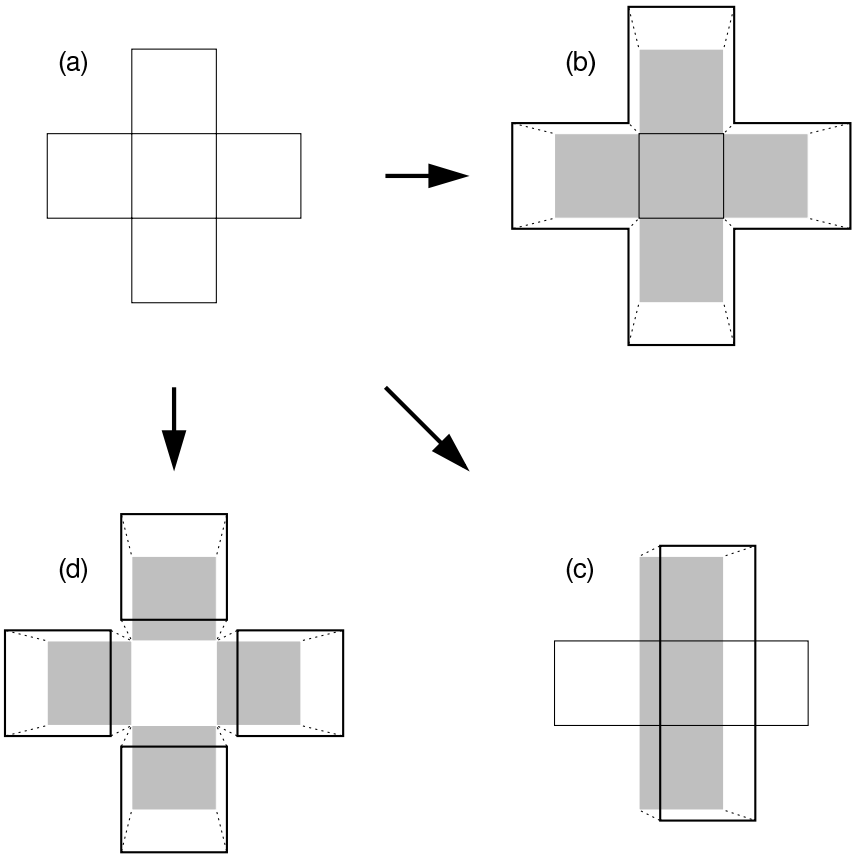


Figure 4: Three decompositions of a one shape

2.2 The Structure of Shapes

Even fairly simple shapes can be composed and therefore decomposed in different and often incompatible ways. To take one example, figure 4 shows a simple compound, a cross shape, decomposed into sub-shapes in three different ways (there are other ways of decomposing it; these three were simply chosen to illustrate the point). These three means of decomposing the original are structurally incompatible with each other. In particular, given a drawing that breaks the figure up in one of these ways, and another drawing that breaks it up in another of these ways, an agent cannot compare the drawings and see that the same shape is present in both unless it is somehow able to *re*-represent the two figures in a structurally compatible manner—or else the shape representation *itself* is part of the comparison between the drawings.

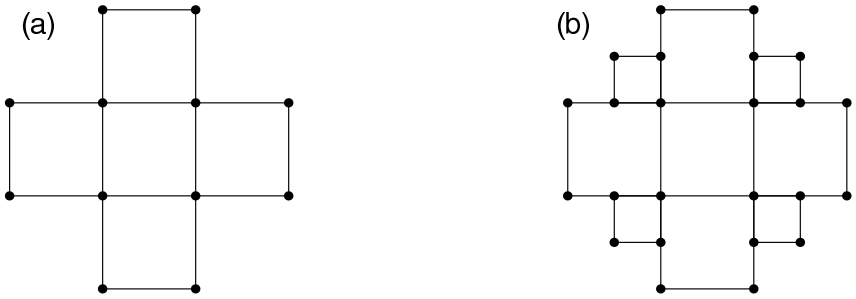


Figure 5: Shape as a plane graph: the second figure should but does not contain the first

2.2.1 Composition and Decomposition

When the input data are all line segments, the most basic pieces of information to consider are the intersections of those line segments. If one takes the cross shape from figure 4(a) and represents all line segments from line intersection to line intersection—that is, breaking line segments at all intersection points so each line segment consists only of its immediate endpoints—one gets a plane graph (in the graph-theoretic sense[2]), as shown in figure 5(a). Figure 5(b) shows the plane graph of a figure that contains our original within it, but the resulting graph structures are not compatible. Geometrically and visually (modulo the dots used to emphasize the line intersections), the shape in figure 5(a) is in fact within that of 5(b) and so a visual reasoner should be able to discover the one pattern within the other figure. What is needed is a way of making their representations compatible.

2.2.1.1 Line Intersections

The driving concern of this investigation, as with all work on knowledge representation, is that *implicit information is computationally useless, and must be made explicit if it is to be useful*. In these examples, the most basic facts have to do with *collinearity* (and, as discussed later, co-circularity) and *line intersections*. Thus, one very simple step is to group the edges that are collinear. That is, one takes all paths (even trivial paths of one edge) along which every edge is collinear to the previous edge. This gives a grouping shown in figure 6. These are the *straight edge paths* from the figure.

At this level it becomes possible to compare the figures. In particular, each of the straight edge

[2] A plan*ar* graph is a graph that can be embedded in the real plane $\mathbb{R}^2$, whereas a *plane* graph is an actual planar embedding of a graph; the distinction is important as any planar graph may have several possible planar embeddings, but here we may wish to talk about the coordinates of actual edges and vertices in an actual planar embedding.

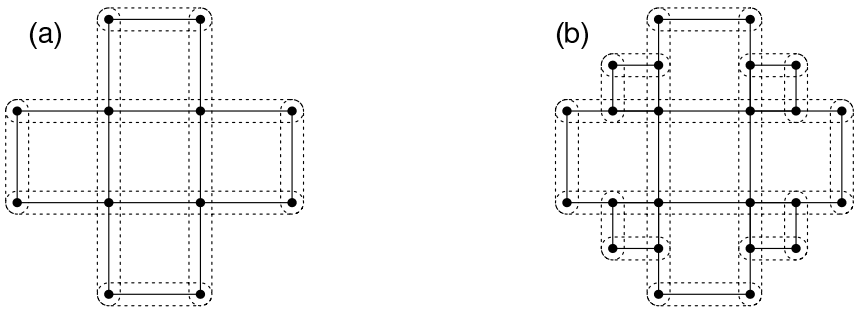


Figure 6: An illustration of straight edge paths

.

paths from the graph in figure 6(a) can be found in the graph in figure 6(b), and most importantly they intersect each other in the same ways: four horizontal and four vertical. This can be illustrated with a new graph structure: for each of the straight edge paths, add a vertex to the new graph, connecting any pair of vertices with an (undirected) edge whenever the associated straight edge paths (or, more precisely, the line segments corresponding to them) intersect in any way. For a given plane graph G of a figure, I will call this the *line intersection graph* $\mathfrak{L}(G)$.

This combines two separate graph theoretic notions. An *intersection graph* for a family of sets $\{S_i\}$ takes a vertex v_i for each set S_i and an edge between two vertices v_i, v_j whenever the associated sets intersect, $S_i \cap S_j \neq \varnothing$. Here, our sets are line segments treated as point sets in the plane. A separate notion is that of a *line graph*. In graph theory, the line graph $L(G)$ of a graph G has a vertex for each edge in G and connects these new vertices in L whenever the associated edges in G are incident. The structure I'm describing, the line *intersection* graph, is similar to this but not exactly the same, since I'm taking *groups* of edges in G with particular properties (namely, being incident and collinear) and mapping these edge groups to single vertices in the intersection graph $\mathfrak{L}(G)$.

Figure 7 shows the line intersection graphs associated with our example figures. Most importantly, the intersection graph of the first figure is now a proper subgraph of the intersection graph of the second. Each edge in the intersection graph corresponds with a line crossing in the original, a vertex in the plane graph. Thus, a particular subgraph isomorphism between two intersection graphs such as this pair will inform a mapping between the line intersections of the two original figures.

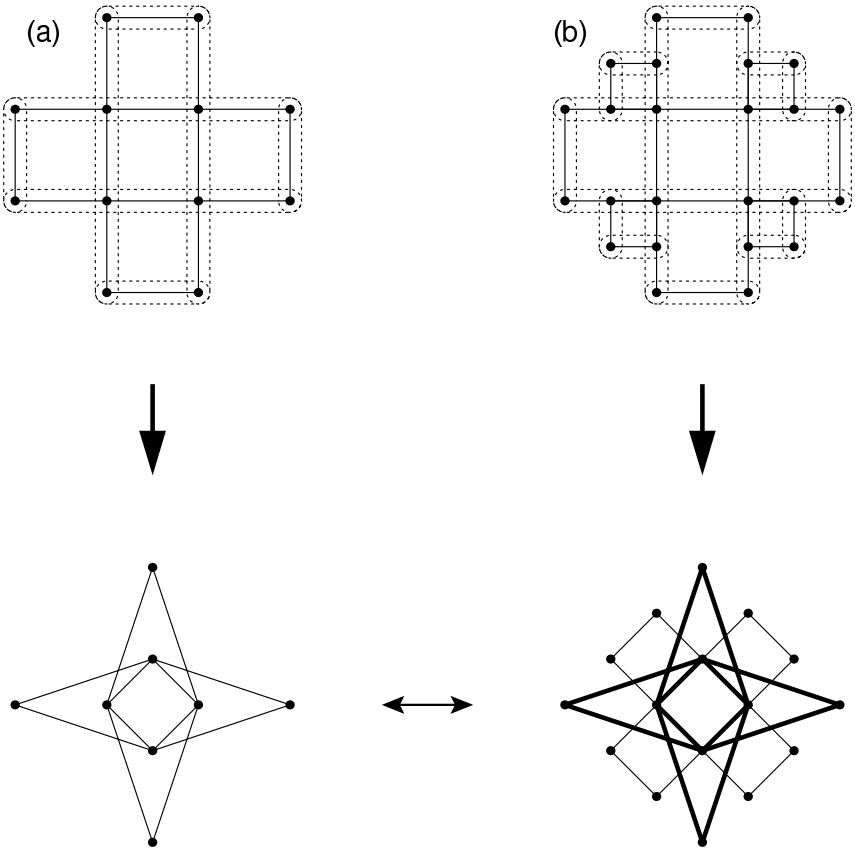


Figure 7: The line intersection graphs associated with figure 6

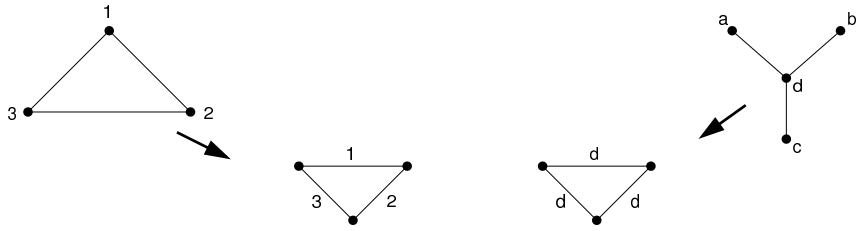


Figure 8: Ambiguous line intersection graphs

2.2.1.2 Edge Labelling

The graphs as I've described them so far contain some important ambiguities which this example doesn't show. First and foremost, a square will map to a rectangle, a rhombus, a trapezoid, or any quadrilateral whatsoever: line length and angle of intersection are not taken into account here. However, there is a more interesting problem that is somewhat more subtle: a triangle and a crossing of three lines at a point will *both look like the 3-cycle* C^3 in their line intersection graphs, as shown in figure 8. More generally, lines intersecting in a polygon and lines intersecting all at a point each look very similar to the other in their line intersection graphs.

Here, two completely different figures have nevertheless produced identical line intersection graphs. The explanation is straightforward: each intersection between a *pair* of lines is a separate edge the intersection graph, so when multiple lines intersect at a point this will make for a complete subgraph of the intersection graph—that is, k intersecting lines in a figure have a line intersection graph that is a k-clique, which will contain within it, for example, any n-gon for $n \leq k$. The way out of this can be seen already in figure 8: if I label edges in the intersection graph by the intersection they represent, then a true isomorphism should associate distinctly labelled edges with distinctly labelled edges and, likewise, identically labelled edges with identically labelled edges. Thus, the pair in figure 8 would not match as three distinctly labelled edges would have to map onto three identically labelled edges.

That edges in $\mathfrak{L}(G)$ correspond to vertices in G, and in particular vertices that form the intersection between two straight edge paths, is important. If there is a subgraph isomorphism m from $\mathfrak{L}(G)$ to $\mathfrak{L}(H)$, then this mapping connects vertices and edges with vertices and edges in a one-to-one manner. A vertex correspondence in m will imply a correspondence between straight edge paths in G and H, and an edge correspondence will imply a correspondence between vertices in G and

H—vertices that may not be adjacent in H but will be always connected along a straight edge path.

2.2.1.3 Length and Angle

The second ambiguity in these line intersection graphs is that of angle and length. In particular, given a common cycle between two line intersection graphs, where each edge on the cycle in each graph has a distinct label, the angles between adjacent edges is not taken into account and neither is the relative length of any edge. Thus, one might map a square onto a rhombus, a parallelogram, a rectangle, a trapezoid, or an irregular quadrilateral: all are 4-sided figures with 4-cycles for their line intersection graphs.

One way out is to construct a label on each node in the line intersection graph indicating the relative length along that segment between a pair of endpoints. This would have to be a complex label, however, since the relevant information is the length between a pair of points represented by incident edges, not the overall length of the line segment. As for angle, that would have to be part of the edge label. Recall from above that an edge in the intersection graph can be labelled with the vertex in the plane graph that it corresponds to. It not only corresponds to a vertex, but an intersection point between two line segments, and so I can also annotate that edge in the intersection graph with the relative angle between the line segments. All this adds complexity to the comparison process, however, and opens up problems of scale and degree of difference. As such, this problem remains future work, which will be discussed at the end. The one concession that Archytas makes is to add a flag to each intersection indicating whether whether or not the intersection is between perpendicular lines; line length is not represented at all.

2.2.1.4 Circles and Arcs

The extension of straight edge paths to circular arcs is straightforward. Just as two line segments are collinear precisely when they are both contained within the same geometric line, two arcs are co-circular precisely when they are both contained within the boundary of the same geometric circle. An arc at a different radius or center point would then have to be part of a different edge path. From this, a new set of edge paths can be constructed, the *straight* as well as *curved edge paths*. However, line intersection graphs get more complicated when circles and arcs are added. In particular, a straight line can only intersect another straight line at a single point, but a circular arc can intersect

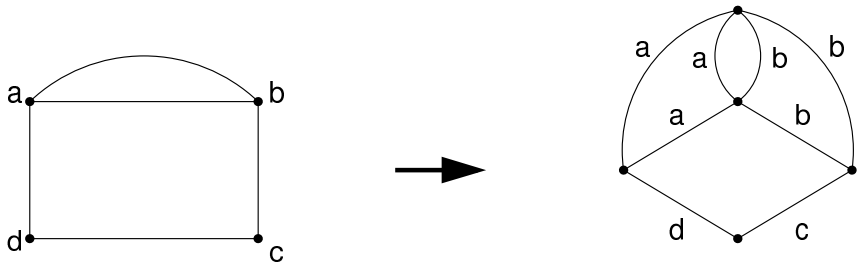


Figure 9: Line intersection graphs of a shape with a curved edge

a line or another circular arc as much as twice. This implies that the resulting line intersection graph may have vertices with two edges between them, corresponding to two intersections. It thus becomes a *multigraph*, and the edge labels serve to distinguish particular intersections between a pair of curves or a line and a curve from each other.

Figure 9 shows an example. Here, the curved "dome" over the top of rectangle *abcd* intersects line $\overline{ab}$ at two different points, *a* and *b*, and so the line intersection graph has two edges going between those vertices, differently labelled. Notice also the two "triangles" (that is, 3-cycles) in the intersection graph—the *aaa* cycle and the *bbb* cycle—that each correspond in fact to three lines/curves meeting at a point, rather than real triangles. In addition to this, there are two *abcd* "rectangles" in the line intersection graph: one corresponding to the actual rectangle *abcd* and the other replacing the top edge with the curved arc. These both appear as 4-cycles in the intersection graph.

Another example is given by figure 10. In this example, the one conch-shell-shaped curve is actually made up of two circular arcs with different radii, and so they have two different vertices in the intersection graph. This example also exhibits the multiple intersection property of curves and lines as well as an arc path of constant curvature represented by a single vertex, the arc running from *a* through *b* to *c*. Note also that the arc wedge $\sphericalangle eac$ looks like a triangle in the intersection graph: for this reason Archytas labels edge sets with a type: line set, arc set, or circle.

Thus augmented with (1) perpendicularity (discussed above in §2.2.1.3) and (2) edge set type (straight line, circle, or circular arc), the resulting structure I call the *augmented line intersection graph*. Shapes will thus be represented as subgraphs of this structure.

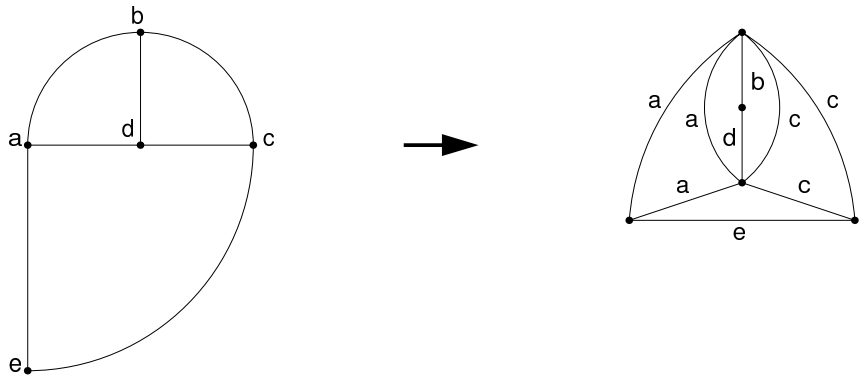


Figure 10: A curved compound shape demonstrating the idea of curved line paths

2.2.2 Definitions

For the sake of concreteness I will here give some mathematical defintions of the above concepts. In particular, chapter 4 will make use of these definitions in describing the algorithms for inferring a shape analogy between source and target.

2.2.2.1 Plane Graphs

The input will be of the form of a line drawing, and that line drawing will be turned into a (not necessarily connected) plane graph. Thus, I begin with plane graphs. To repeat the standard definition [20, ch. 4], a plane graph $G = (V, E)$ is a pair of *vertices* and *edges*, where:

- Each vertex $v \in V$ is a point in the Euclidean plane, $v = (x, y) \in \mathbb{R}^2$
- Each edge $e \in E$ between a pair of vertices $u, v \in V$ is a continuous path between u and v, i.e. there exists a continuous mapping $\varphi : [0,1] \to e \subseteq \mathbb{R}^2$ with $\varphi(0) = u$ and $\varphi(1) = v$
- No two edges cross, i.e. for all edges $e, f \in E$, $e \neq f$, the "interior" of each path, the open path that is the edge minus its endpoints $e^\circ = e \setminus \{u, v\}$, is disjoint from the interior of the other, $e^\circ \cap f^\circ = \varnothing$

Note that a *planar* graph is any graph that is isomorphic to a *plane* graph, the former being a graph constructed in any suitable manner and the latter being a set of points in the plane as just described.

In order to transform an input line drawing into a proper plane graph in this way, it is necessary to compute all of the intersections among all of the line segments (and circles and circular arcs), and

divide all of the line segments (circles, etc.) at these intersection points. This is a straightforward process and well-known algorithms exist that can run in $O(n^2)$ time or better [18, ch. 2].

Although It may be interesting in general to consider more general curves such as splines, I will be dealing with figures consisting only of straight line segments, circles, and circular arcs. Thus, the plane graphs I will deal with have edges that are all either straight line segments or circular arcs. Observe as well that the allowance of arcs means the graphs may be plane *multigraphs* instead of mere plane *graphs* without multiple edges or loops (a loop could occur, e.g., with a line segment terminating on the edge of a circle). This is, of course, perfectly compatible with the definitions just stated.

2.2.2.2 Line Intersection Graphs

A straight edge path P is a subgraph generated by a set of connected edges $e_1, e_2, \ldots, e_k$, $k \geq 1$, such that each pair of edges in the path are collinear. That is, if $v_1, \ldots, v_{k+1}$ are the vertices these edges connect, then $\overline{v_1, \ldots, v_{k+1}}$ forms a single line segment. It is important that these paths be maximal: there is no other edge $e \notin \{e_1, \ldots, e_k\}$ that can extend this line. A curved edge path P is a subgraph generated by a set of connected edges $e_1, \ldots, e_k$, $k \geq 1$, that are all co-circular—that is, there is some circle whose boundary set contains all of them as subsets. Again, such a path is expeceted to be maximal, so that no additional edge can extend it. Finally, any straight or curved edge path may of course be a trivial path consisting of a single edge.

Let $\boldsymbol{P}$ be the set of all maximal straight edge and curved edge paths in G including trivial single-edge paths (I will more often call them *line sets* and *arc sets*). The *line intersection graph* of a plane graph G is the (abstract, non-plane) graph $\mathfrak{L}(G) = (V, E)$ where:

- For each path (i.e. line/arc set) $P \in \boldsymbol{P}$ there is a vertex $v_P \in V_{\mathfrak{L}(G)}$
- For any two paths $P, Q \in \boldsymbol{P}$, whenever there is a vertex $v \in V_G$ that is an element of both paths—i.e. the two corresponding lines or curves intersect at vertex (intersection point) v—then there is an edge $e_{v,P,Q} \in E_{\mathfrak{L}(G)}$

Note that if the original graph is not connected, neither will the line intersection graph be connected, and in general the line intersection graph only captures relationships among the connected elements

of a shape, and this is true even within a closed figure. For instance, opposite sides of a quadrilateral will not be related to each other in any way, or, to take a somewhat better example, if two non-adjacent edges in a non-convex polygon happen to be collinear, such as the tops of a U-shaped figure, this fact will not be captured by the line intersection graph; and a pair of rectangles of equal length that are parallel to each other, if they are not connected, will be unconnected (and therefore unrelated) in the line intersection graph as well.

In general a graph isomorphism between two graphs G and H is a mapping $m : G \to H$ such that

1. For $u, v \in V_G$, $u \neq v$ if and only if $m(u) \neq m(v)$
2. For $e, f \in E_G$, $e \neq f$ if and only if $m(e) \neq m(f)$
3. $v \in V_H$ only if $\exists u \in V_G$ with $m(u) = v$
4. $f \in E_H$ only if $\exists e \in E_G$ with $m(e) = f$
5. $e = \{u, v\} \in E_G$ if and only if $m(e) = \{m(u), m(v)\} \in E_H$

The first four conditions simply state the bijective property (that the mapping is one-to-one and onto), but the last condition is critical in that it states that the isomorphism should preserve the structure of the graph: adjacent edges should map to adjacent edges. And finally, of course, a *sub*graph isomorphism from G to H is a graph isomorphism between G and some *sub*graph S of H.

Here we shall define an isomorphism between the line intersection graphs $\mathfrak{L}(G)$ and $\mathfrak{L}(H)$ of two plane graphs G and H as a graph isomorphism $m : \mathfrak{L}(G) \to \mathfrak{L}(H)$ where, for any two edges $e_{u,P,Q}$ and $e_{v,R,S} \in E_{\mathfrak{L}(G)}$ with $m(e_{u,P,Q}) = f_{x,A,B}$ and $m(e_{v,R,S}) = f_{y,C,D}$, then $u \neq v$ in G if and only if $x \neq y$ in H—in other words, it preserves the edge labelling.

2.2.2.3 Geometric Correspondence

Let G and H be two planar graphs, with associated line intersection graphs $L_G = \mathfrak{L}(G)$ and $L_H = \mathfrak{L}(H)$, and let $m : L_G \to L_H$ be an isomorphism between the line intersection graphs. Define a partial mapping $\mu : V_G \to V_H$ where, if $m(e_{v,P,Q}) = f_{x,A,B}$ for $e_{v,P,Q} \in E_{L_G}$ and $f_{x,A,B} \in E_{L_H}$ then let $\mu(v) = x$.

This mapping μ gives us a vertex-to-vertex association between the two drawings, and from this we can reconstruct edge correspondence. If $m(e_{v,P,Q}) = f_{x,A,B}$, $m(v_P) = v_A$, and $m(v_Q) = v_B$—and

thus $\mu(v) = x$—then v is on both P and Q in G and x is on both A and B in H and we can associate the edges in P (in G) with the edges in A (in H), and likewise for the edges in Q and B in G and H, respectively.

Let C_L be a cycle in L_G with distinct edge labels, and likewise D_L be a cycle in L_H with distinct edge labels, with $m(C_L) \mapsto D_L$—that is, the vertices and edges of the one mapping onto the other. Now, let $e_{u,P,Q}$ and $e_{v,Q,R}$ be two incident edges along the cycle C_L with v_Q the common vertex between them. Then we know if $m(e_{u,P,Q}) = f_{x,A,B}$ and $m(e_{v,Q,R}) = f_{y,B,C}$ that $\mu(u) = x$, $\mu(v) = y$, and also that there is a (sub)path along a straight or curved edge path from u to v in G that corresponds to a (sub)path from x to y along a straight or curved edge path in H. For each pair of edges in the line intersection graph (and their corresponding images under m) we can construct independent paths between corresponding vertices in G that map to independent paths in corresponding vertices in H, where the vertices in G map onto those in H via μ. In this way we can reconstruct cycles C in G and D in H where some of the vertices of the one map onto those of the other under μ, and the remainder are along corresponding edge paths and can be disregarded.

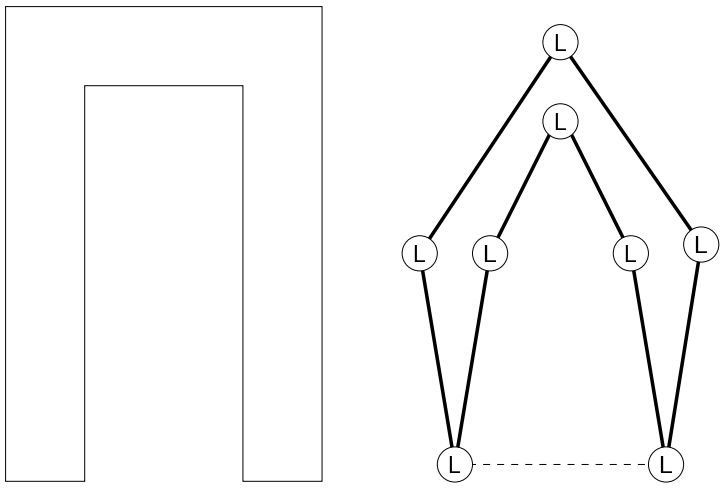
Looking ahead, if C is already known as a shape in G that depicts some component in a structural model, we can hypothesize that D depicts a similar component in H as well. This process can be generalized to any substructure. Here I have considered cycles (which in general correspond to polygons in the original image) but the ideas generalize to any shape pattern.

2.2.3 Additional Information

The representation discussed so far is quite spare, and all of the relationships captured are very local—between connected line segments only. It must be possible to add a bit of knowledge to capture relationships, say, within a shape but not necessarily between connected segments. In this spirit, then, two more pieces of information are represented in the augmented line intersection graphs in addition to those discussed above: disconnected collinear segments, and all the faces or cells bounded by cycles of edges in the original image.

2.2.3.1 Disconnected Collinear Line Segments

The need for representing relationships between disconnected segments that are collinear can be shown better by example. Figure 11(a) shows a simple U-shaped polygon. In the bottom of the



(a) An inverted U-shaped polygon

(b) It's associated line intersection graph, where the dotted lines indicate disconnected collinear segments. L indicates line segment, edge labels have been omitted

Figure 11: An example of the representation of disconnected collinear line segments

figure the two line segments are aligned, but this fact is not captured by the line intersection graph, and so the pattern might match a figure in which the two "legs" are substantially different lengths. However, it is straightforward to add a test whereby two line sets in the line intersection graph are related to each other if they happen to also be collinear. Figure 11(b) illustrates the resulting augmented line intersection graph representation (sans edge labels).

2.2.3.2 Faces

An example will also better illustrate the need for representing when two line segments both bound the same face. A face in a plane graph is a region of the real plane surrounded by edges and vertices of the graph in question. Figure 12 shows two images, three stacked blocks on the left and two stacked blocks on the right, along with their line intersection graphs. We would expect there to be two mappings in this example, where the two blocks map to the top two in the shape on the left and where they map to the bottom two (ignoring rotation and reflection), but in fact there are *four* mappings. The two additional mappings both take line $\overline{uv}$ to $\overline{ab}$ and $\overline{yz}$ to $\overline{gh}$, with $\overline{wx}$ mapping either to line $\overline{cd}$ or line $\overline{ef}$. Here, two adjacent blocks are mapping to two *non*-adjacent blocks.

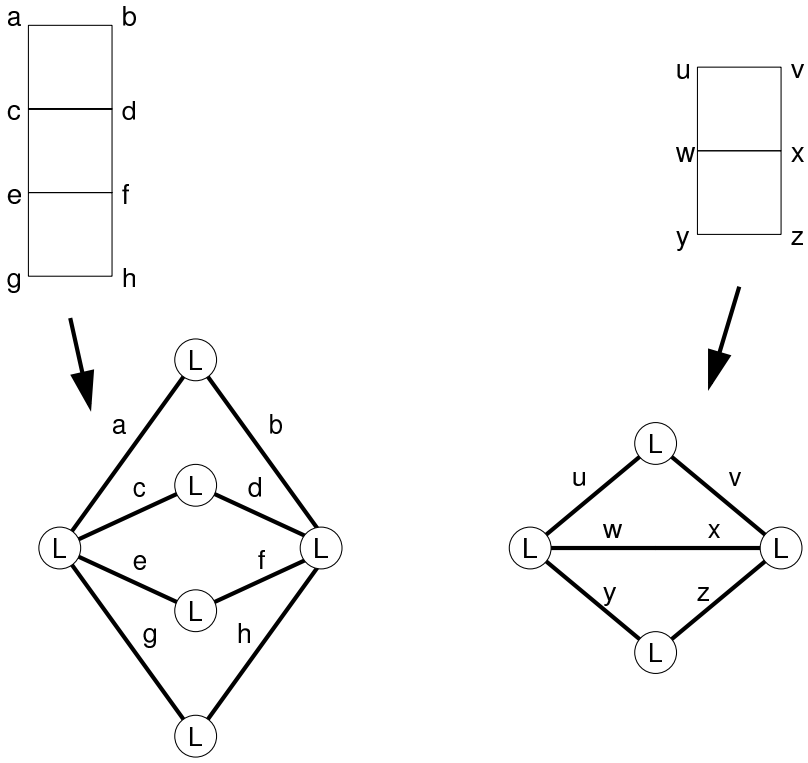


Figure 12: Adjacent blocks may match nonadjacent blocks

Furthermore, the various edges may map to each other in *the wrong order*. For instance we may have $\overline{uv}$ to $\overline{cd}$, $\overline{wx}$ to $\overline{ab}$, and $\overline{yz}$ to $\overline{ef}$. With more complex figures, the unexpected mappings multiply even more. When mapping a relatively simple shape to a complex drawing, often there are many mappings that are even more strange than this, making the recovery of structure from shape even more difficult.

In a plane graph, cycles of edges and vertices bound regions in the plane called *faces* or *cells*. Figure 13 shows the images from figure 5 along with their *planar duals* in grey, minus the outer face (the region surrounding the shape). A planar dual takes a vertex for every bounded face in the original graph and connects with an edge those faces that are separated by an edge.

It is straightforward to produce a list of all such faces when calculating the line intersections [18, ch. 2], where each face is listed by the vertices and edges on its boundary. From this we can add another class of nodes in the augmented line intersection graph, those representing these bounded faces (again minus the outer region surrounding the figure), and connect them to those vertices (line, arc, and circle sets) and edges (intersection points) on their boundaries. This does

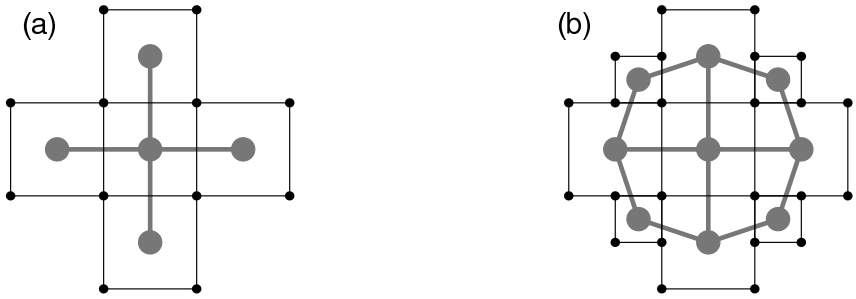


Figure 13: The planar dual, minus the outer face, of the images from figure 5

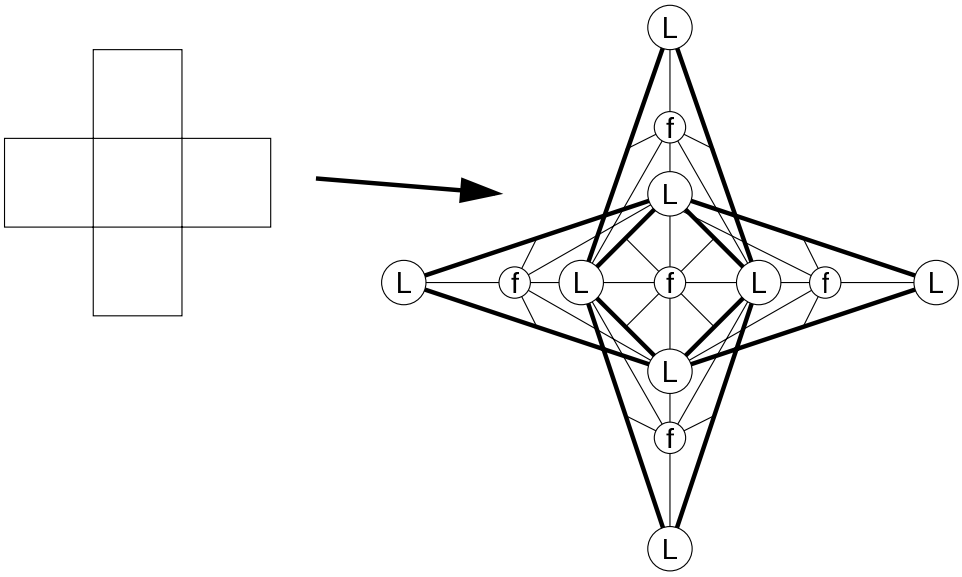


Figure 14: The representation of faces in line intersection graphs, where f indicates a face

not represent the planar dual directly but rather with one level of indirection: two faces are adjacent when their boundaries share an edge. An example of this is shown with the by-now-familiar cross shape in figure 14 and its associated augmented line intersection graph with the face information.

In terms of notation, the faces of a graph G are denoted $F(G)$. Note that each face $f \in G$ is a point set in the real plane. The frontier or boundary of a face is the set of edges and vertices surrounding it. If the boundary of a face is the set X then an edge is on the boundary when $e \subseteq X$, otherwise $e \cap X = \varnothing$ [20, ch. 4]. I will write $E_{\mathfrak{L}(G)}(f)$ and $V_{\mathfrak{L}(G)}(f)$ for the set of edges (intersection points) and vertices (line/arc sets) on the boundary of a given face $f \in F(G)$.

2.2.4 Tradeoffs

The augmented line intersection graph is analogous in many ways to the representation used in GeoRep's low-level relational describer (the LLRD), and a comparison is useful. In some sense the two approaches are at opposite ends of a spectrum: the line intersection graphs represent a minimal amount of information, adding new pieces of knowledge a bit at a time only as necessary, and thus explore how far one can get on a certain kind of representation with no further criteria for a match. GeoRep, by contrast, has routines for many kinds of relationships, not just connectivity and perpendicularity, but also concavity along the sides of polygons, parallelness, proximity, relative length, and so on. GeoRep associates not just connected segments, but also proximate ones (using a simple heuristic to determine proximity), and then performs additional tests on proximate pairs of segments (such as relative length, parallelness, and so on). Compared to these line intersection graphs, the relational structure of GeoRep's LLRD representation is, in general, more dense and with a great deal more information.

The tradeoff between the comparatively spare augmented line intersection graphs and the much more expressive representation used in GeoReps LLRD is that, for the matching methods that are used in Archytas (described in chapter 4), the more information is placed into the representation, the more complex the matching process will be. It remains an open empirical question for future work as to how much of the information represented in GeoRep's LLRD can be added to the line intersection graphs while still keeping the shape matching efficient (or, conversely, whether the methods described here and in chapter 4 can be incorporated into a more elaborate diagrammatic reasoning system such as GeoRep).

2.3 Models of Drawing Content

The structure, shape, and drawing in a source case form an abstraction hierarchy. The structural model is a specification of components, quantities, and structural relations, along with properties (height, width, etc.) and variable parameters (e.g. position, angle) of each. Figure 15 illustrates the graph representing the connections among the components in the structural model of the piston and crankshaft device in figure 1(a) (page 2). In addition, for each component, the user specifies the coordinates of a *basic shape* corresponding to that component, and for each structural relation

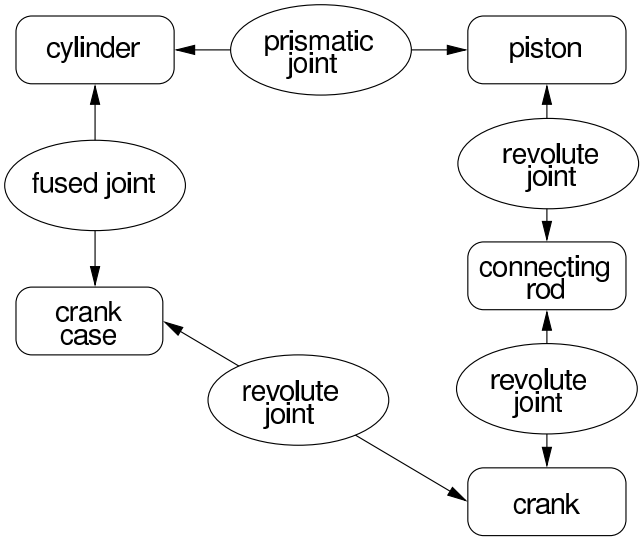


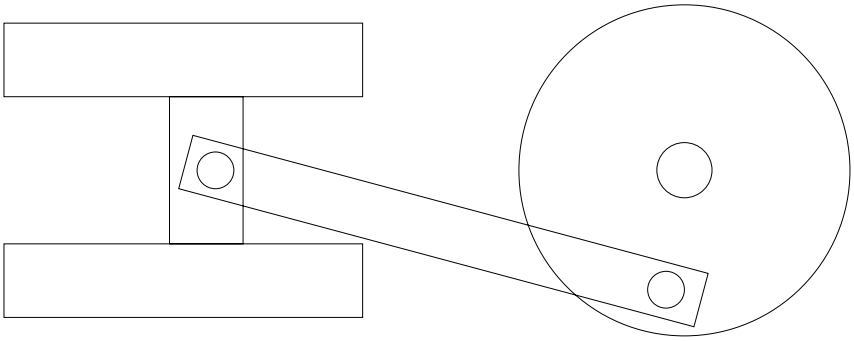
Figure 15: Structural model for piston and crankshaft

(or connection), the user specifies a *composite shape* as a composition of basic shapes depicting the components involved in the connection. This produces the drawing/shape/structure portion of the abstraction hierarchy shown in figure 2 (page 6).

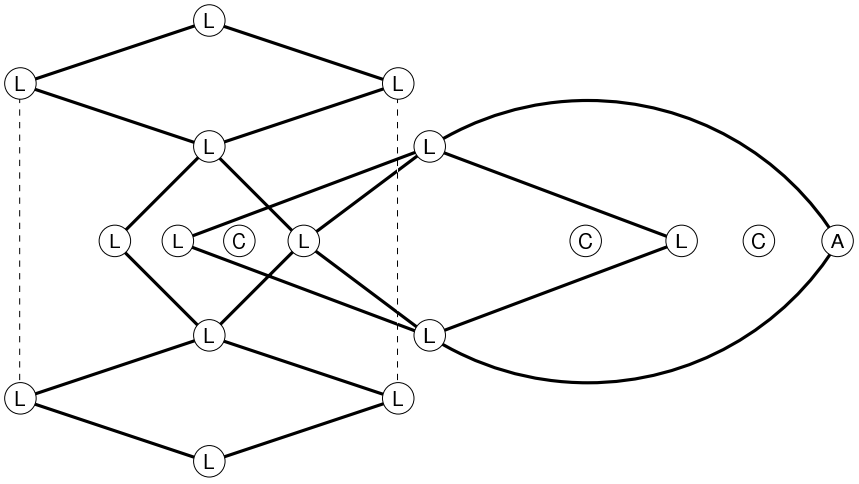
Archytas first reads in a drawing; fill patterns and layering are ignored, so after pre-processing, the input drawings to Archytas look like figure 16(a). Line segments are taken as maximally connected collinear segments, and likewise for co-circular connected arcs, while Archytas calculates all the intersection points between them. These elements form the vertices V of the line intersection graph, and the intersection points the labelled edges. Figure 16(b) shows the line intersection graph of the piston and crankshaft example.

2.3.1 Basic and Composite Shapes

Depending on the perspective of the drawing, a particular component may be depicted by an actual shape, which is a subgraph of the line intersection graph, or not at all. Figure 17 shows the basic and composite shape hierarchy for the piston and crankshaft example (the drawing itself is shown rather than the intersection graph). Each component in the model shown in figure 15 except for the crankcase is linked to a basic shape, and each structural relation except for those involving the crankcase is linked to a composite shape. As illustrated in figure 18, the two-level basic vs. composite shape hierarchy captures the depicted structure of the components in the device. The



(a) The piston and crankshaft drawing from figure 1(a) with fill patterns and layering ignored as they are in Archytas



(b) The augmented line intersection graph of this image, where L indicates a line set, C a circle, and A a circular arc set

Figure 16: The augmented line intersection graph of the piston and crankshaft

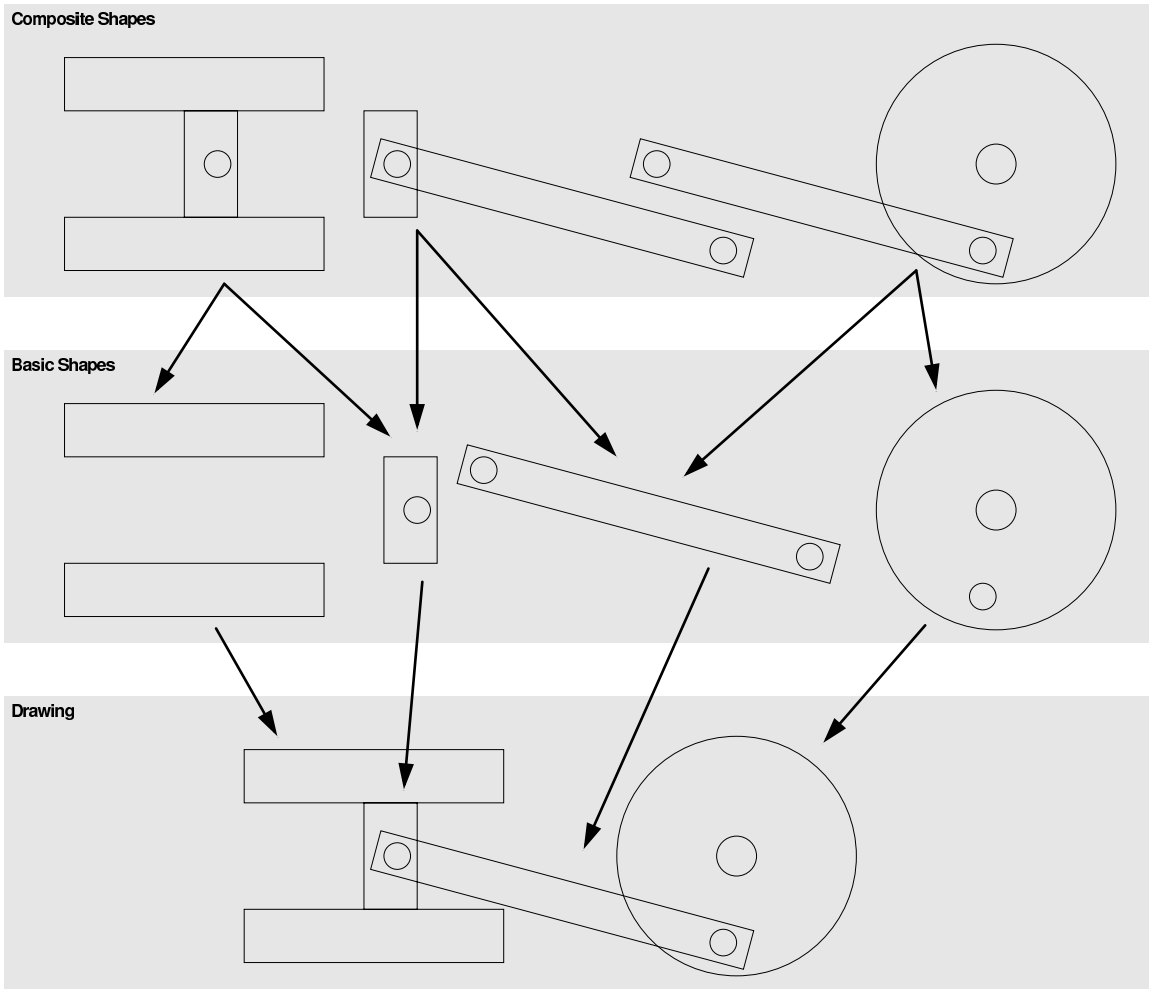


Figure 17: The breakdown of basic and composite shapes in figure 16

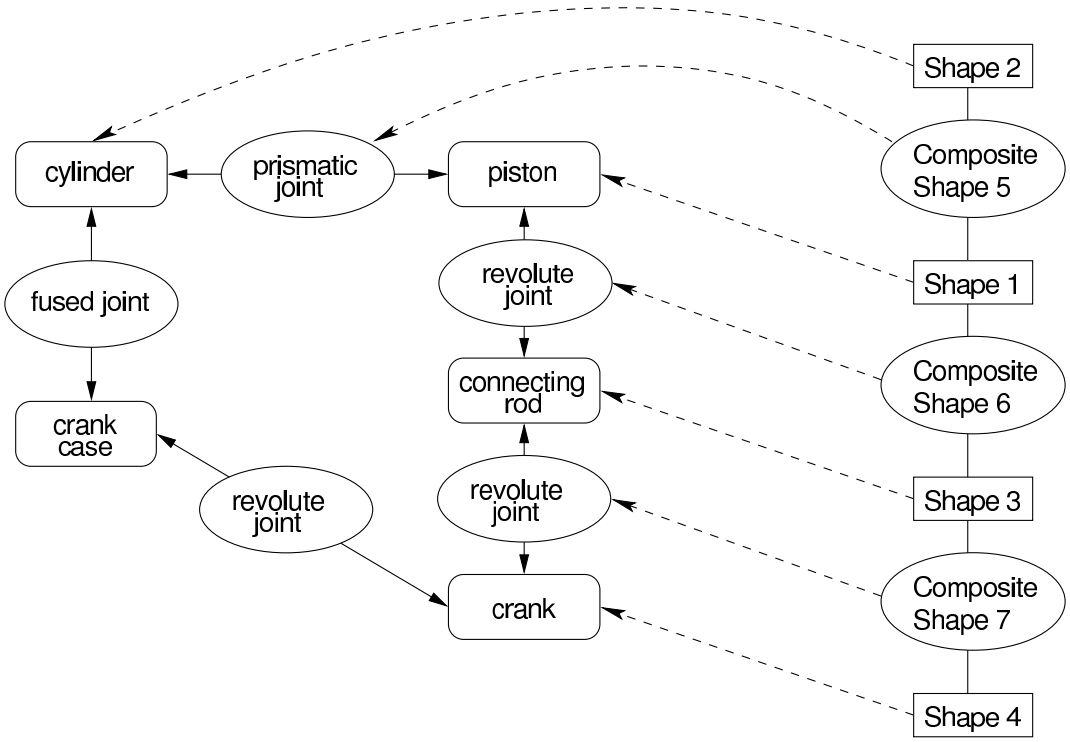


Figure 18: The correspondence between shape and structure

analogical method, discussed in chapter 4, attempts to map these shape patterns to the target drawing and then transfer first whole shapes and then structural elements, building up mappings along the way. The representation discussed here provides for a basis on which that mapping process can proceed.

With each depicted component in the DSSBF model is associated a basic shape. That shape is a specification of points, lines, intersections, arcs, circles. That is, it is a subgraph $B \subseteq \mathfrak{L}(G)$ of the augmented line intersection graph. The complete set of basic shapes in the drawing B is a family of such subgraphs, $B = \{B_i\}$, where $B_i \subseteq \mathfrak{L}(G)$. A composite shape is associated with all depicted structural relations in a DSSBF model, and is specified simply in terms of the basic shapes of the components involved in the relation. Thus, if two connected components are depicted by basic shapes B_i and B_j, then the structural relation between them is depicted by the induced subgraph $\mathfrak{L}(G)[V_{B_i} \cup V_{B_j}]$—that is, the circles and line and arc sets are gathered together from both shapes and all intersections between them are determined. The set of all composite shapes in a drawing forms a family of subgraphs $C = \{C_i\}$, where once again $C_i \subseteq \mathfrak{L}(G)$, and of course for each C_i there are two basic shapes $B_j, B_k \subseteq C_i$.

CHAPTER III

ORGANIZATION OF THE DEVICE MODELS

Drawings depict devices, and so the other half of the representational problem faced in this work is to model the devices themselves. The models employed in this work are *Structure Behavior Function* (SBF) models of devices [47, 12, 6]. This chapter reviews the organization of SBF models.

An SBF model of a device specifies the manner in which the structural elements of the device interact to produce the device's output behaviors, or function. Thus, the knowledge captured is *structural*, *causal*, and *teleological*. It is structural because it represents the structural elements of the device: components, quantities (what were called "substances" in earlier work), and structural relations between them, as well as the properties of these elements. It is causal because it represents qualitative states of components and quantities and transitions between these states, and in particular because state transitions, which represent change, are related to their immediate physical causes. It is teleological because it represents the output behavior of a device *as distinct from* the internal behaviors of a device, this being an explicit representation of function or purpose. These parts of the model are all described below.

In the discussion that follows, I will present detailed examples to lend some concreteness to the discussion, but more importantly as a presentation of the test data employed in evaluating the work. The data set compiled for testing Archytas comprised 26 different drawings across three domains (including source drawings), that of piston and crankshaft devices, door latch mechanisms, and pulleys. Although all of these drawings were run, a detailed evaluation was done on a particular selection of cases, and so these are the cases I present here. There were five source cases used to run these examples across the three domains, one from the piston and crankshaft domain, one from the doorlatch domain, and three from the pulley domain. I will describe the first of the pulley examples here, the other two are modifications of this example and will be presented in chapter 6.

Figure 19: The Structure to Function Mapping

3.1 Background

The Kritik system [47, 46] was an autonomous case-based design system that addressed the task of preliminary (conceptual, qualitative) design of physical devices. It took as input a specification of the desired function of a device, and gave as output a specification of its structure. Since Kritik addressed the $F \mapsto S$ (function to structure) design task, its design cases contained an inverse $S \mapsto B \mapsto F$ (structure to behavior to function) mapping of known designs, where the B in a $S \mapsto B \mapsto F$ mapping stood for the internal causal behaviors that composed the functions of the components in the design into the functions of the design as a whole (figure 19). Kritik's SBF model of a design represented function and behavior at multiple levels of aggregation and abstraction, and organized them into an $F \mapsto B \mapsto F \mapsto B \mapsto \cdots \mapsto B \mapsto F(S)$ hierarchy. Kritik showed that the SBF models provided a vocabulary for representing, organizing, and indexing design cases, methods for retrieving and adapting known designs to meet new (but related and similar) functions, and methods for verifying and storing new designs in memory.

The origin of Krikik's SBF models lies in Chandrasekaran's Function Representation (FR) scheme for representing the functioning of devices [68, 13]. Similar ideas have been developed in other areas, however. In cognitive engineering, Rasmussen [66] developed similar SBF models for aiding humans in trouble-shooting complex physical systems. In computer-aided engineering, Tomiyama developed similar FBS models [81] for aiding humans in designing mechanical systems. In design cognition, Gero [39] developed similar FBS models for understanding the mental information processing of designers in general. In their analysis of verbal protocols of design engineers working in a variety of domains, Gero and McNeill [38] found that while novice designers spend most of their time on the structure of the design solutions, spending relatively little time on the design functions or behaviors, expert designers spent significant amounts of time on all three major elements of FBS models: function, behavior, and structure.

Table 1: A component schema in SBF

Slot	Type	Number
Component Name	*string*	1
Properties	*property*	0+
Parameters	*parameter*	0+
Primitive Functions	*component function*	0+
Structural Relations	*connection*	0+
Shape	*basic shape*	1

Table 2: A quantity schema in SBF

Slot	Type	Number
Quantity Name	*string*	1
Properties	*property*	0+
Parameters	*parameter*	0+
Structural Relations	*quantity relation*	0+

3.2 Structural Model

The last chapter discussed the shape model in terms of basic and composite shapes, which depict components and structural relations, respectively. Quantities in this model are not regarded as being depicted by shapes, since the only quantities in this application, kinematics devices, are abstract quantities such as linear and rotational motion. The structural model details the properties of components, quantities, and structural relations between them.

A component is a physical entity in the device possibly serving some primitive functions (these are discussed in more detail below). Components have properties as well as variable parameters in addition to these primitive functions. The user, in specifying a component in a source model, must therefore specify each of these elements. This schema is illustrated in table 1.

Quantities SBF represent such diverse elements as water, heat, linear or rotational motion. In my application, the only quantities present will be linear and rotational motion. Quantities are the objects of primitive functions, discussed below, and have properties and parameters but no shapes or primitive functions (since they are only the objects of primitive functions they do not themselves serve functions). The schema for a quantity in SBF is shown in table 2.

Table 3: The schema for a property of a component or quantity

Slot	Type	Number
Property Name	*string*	1
Property Type	*scalar*, *vector*, or *angle*	1
Component/Quantity	*component* or *quantity*	1
Value	*number*	1

Table 4: The schema for a structural relation

Slot	Type	Number
Relation	*symbol*	1
First Argument	*component*	1
Second Argument	*component* or *quantity*	1
Shape	*composite shape*	0–1

Both components and quantities take properties and parameters. Here, the only difference between a property and a parameter is change: a property has a single static value, whereas a parameter can vary. Properties and parameters possess a type, being either scalar, vector, or an angle. Properties have a value and parameters have a range of possible values. These schema for a property is shown in 3; a parameter replaces the *value* slot with a *range* slot but is otherwise identical.

Structural relations come in three varieties: those between components, those between a component and a quantity, and those between quantities. In the domain of kinematics devices there will be no need for structural relations among quantities (an example would be water containing heat), and so we have only two kinds. Relations between components I call *connections* and these may be depicted by a composite shape in the drawing. *Quantity relations*, those between a component and a quantity, are not depicted in a drawing. All structural relations are binary. The schema for a structural relation is shown in table 4.

The most critical slot in the structural relation schema is that of the relation itself. Relations between components can be of several types [48, §3.3]. The relation types employed in this work are the following:

fused two components are fused together when they are affixed or adjoined in some way, and so do not move with respect to each other

revolute joint a connection with one degree of freedom, that of rotation about some axis; in general one or the other of the components may be fixed in position, but this is not necessarily the case; a hinge or pivot is a revolute joint

prismatic joint a connection with one degree of freedom, that of translation about some axis; a bolt in a door latch or a slider along a track are examples of prismatic joints

cylindrical joint a connection with two degrees of freedom, that of translation about some axis and that of rotation about the *same* axis; a piston moving in a cylinder is an example of cylindrical joint

rolling joint when one component rolls over another, such as a rope that is wound around a wheel and rolls over the surface of it as the wheel turns

hooked when one component hooks or touches another and pushes or slides it along in some direction; the cam in a door latch hooks with the shaft to pull back the bolt in the door

These will be seen in detailed examples that follow. The only quantity relation employed in this work is containment: a component **contains** some quantity.

3.2.1 Piston and Crankshaft Example

The piston and crankshaft example was the first one presented, in figure 1(a) (page 2). This is a simple example of a one piston engine as might be seen in, for instance, a lawn mower. Multi-piston engines are of course common in cars, motorcycles, and small airplanes. The device consists of a piston that moves back and forth within a cylinder, and is attached to a connecting rod that connects it to a crankshaft. This crankshaft turns within the crankcase, which is fused to the cylinder.[1] The function of the device is to turn the crankshaft, and it is the forcing of the piston down the cylinder by an external stimulus (namely, the burning of a compressed fuel/air mixture) that produces this behavior.

The structural model has five components: the cylinder, piston, connecting rod, crankshaft, and crankcase. It also has five connections among these components:

[1] In most car engines, the cylinder and crankcase are cast in one block of metal; in some other engines, such as in some motorcycles, the cylinder is a separate component bolted to the crankcase.

Table 5: An outline of the structural model of the piston and crankshaft example

Component	Properties	Connected to
Piston	height, diameter	cylinder, connecting rod
Crankshaft	diameter	crankcase, connecting rod
Connecting Rod	length	crankshaft, piston
Cylinder	diameter, length	piston, crankcase
Crankcase		cylinder, crankshaft

Table 6: An example of a component schema for a piston

Slot	Value
Component Name	piston
Properties	height, diameter
Parameters	—
Primitive Functions	allow(piston-motion)
Structural Relations	revolute-joint(piston, connecting-rod), cylindrical-joint(cylinder,piston)
Shape	piston-rectangle

- cylindrical joint between piston and cylinder
- revolute joint between piston and connecting rod
- revolute joint between connecting rod and crankshaft
- revolute joint between crankshaft and crankcase
- crankcase fused to the cylinder

The first three of these are depicted in figure 17 (page 38), going left to right across the top. The components shown (going left to right across the middle) are the cylinder, piston, connecting rod, and crankshaft. The layout of the connections is shown in figure 15 (page 36). The structural model is outlined in table 5. An example of the detailed slot values for the component schema for the piston is shown in table 6.

What is not shown in the structural model in table 5 are the primitive functions and the quantities contained by the components (an example can be seen in table 6), although primitive functions are specified along with their associated components in the structural model. In the piston and

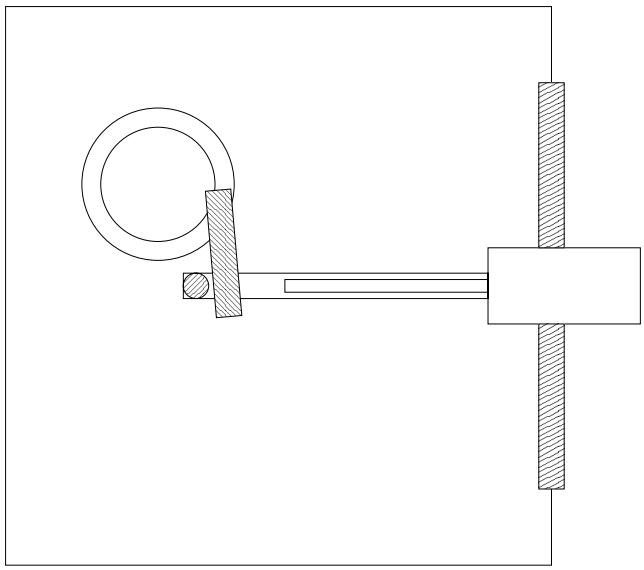


Figure 20: A door latch example

crankshaft example, the quantities are: the linear motion of the piston, both the linear and angular motion of the connecting rod (two different quantities), and the angular motion of the crankshaft. The connecting rod deserves some explanation: this component has two separate motions by virtue of it's being connected to two moving components, the linear motion it gets from moving back and forth with the piston, and the angular motion it gets from being attached at the other end to the crankshaft (this motion, if one were standing on the piston, would look like the back-and-fourth motion of a pendulum). Each component has a primitive function that is to *allow* the relevant quantity.

3.2.2 Door Latch Example

The second example source is a simple door latch mechanism, shown in figure 20. In this device, an external doorhandle (which is not part of the model) turns what is called the "cam", which hooks a peg on a shaft. When the cam turns it pulls back on the shaft and thereby pulls back on the bolt, which is pulled back through a hole in the door.

There are, then, four components: (1) the door, (2) the cam, (3) the shaft, and (4) the bolt. The door is the outlining shape and the hole through which the bolt slides. The bolt is the rectangle sliding through the door. The shaft is the longer, narrower rectangular shape leading back from it, and the cam is a round shape with a rectangular shape protruding from it and overlapping the shaft.

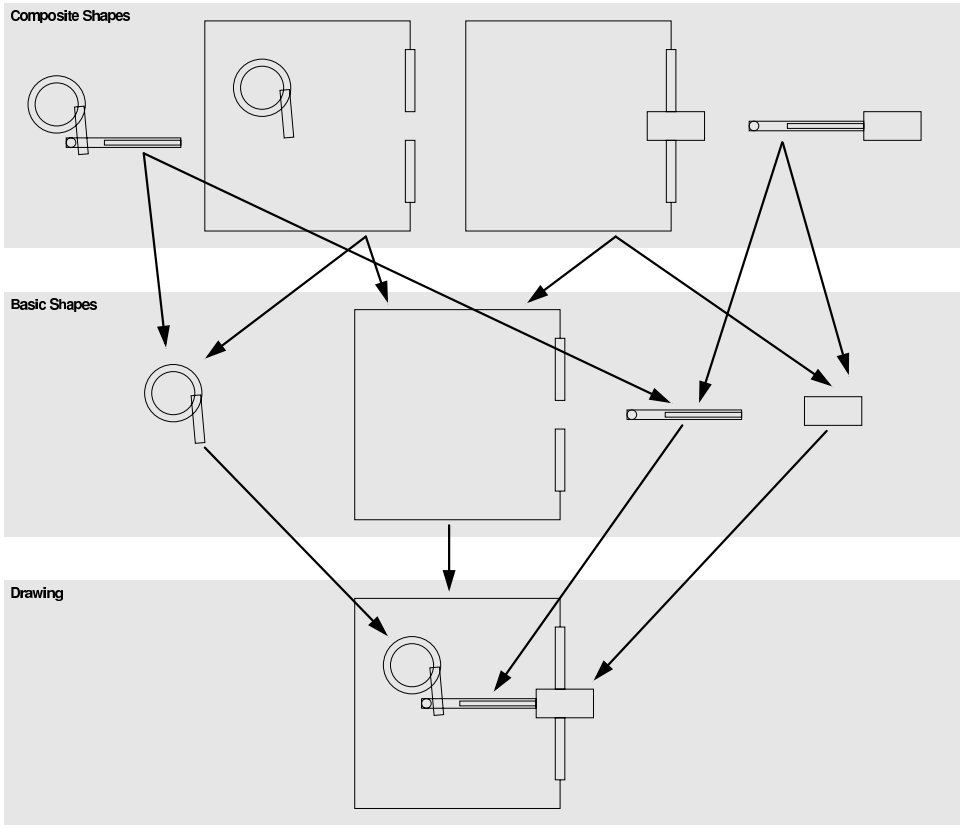


Figure 21: The shape hierarchy in the door latch drawing

There are four structural relations in this model among these components:

- The revolute joint between the door and the cam
- The shaft fused to the bolt
- The cam hooking the shaft
- The prismatic joint between the bolt and the door

The composite shapes for each of these are the combinations of the basic shapes for the participating components (the complete basic and composite shape breakdown for this drawing is shown in figure 21, and figure 22 shows the structure). There are no quantities in this model as there is no continuing motion and hence all the behaviors are component behaviors. As such, there are no primitive functions, either.

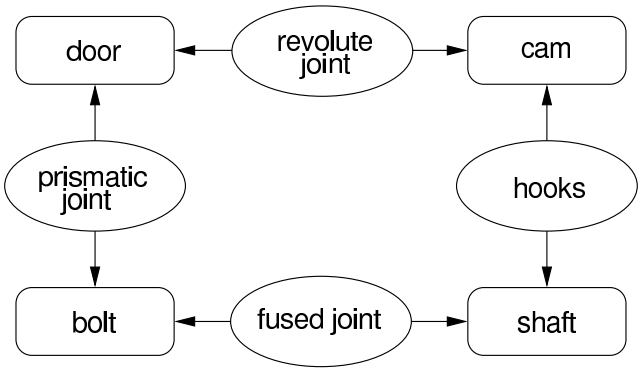


Figure 22: Structural model for the door latch

3.2.3 Pulley Example

The third class of examples comes from the work of Mary Hegarty [50]. Hegarty has investigated the strategies that human subjects use to infer the kinematics of simple mechanical systems, and in particular pulley systems, from diagrams showing their static configurations. Using reaction time and exe fixation data she proposes that, when asked simple questions about the effects that certain actions might cause, subjects attempt to mentally animate the device by decomposing the representation into smaller units corresponding to the device's components and then animating those components. She also shows that subjects have difficulty making inferences against the chain of causality (i.e. from effects to possible causes), and have a preference for inferences in the direction of causation. This proposal is, at an abstract level, remarkably consistent to the methods being described here for a similar task. Although Hegarty does not discuss analogical reasoning, it is possible to make use of some of her examples for Archytas.

A simple two-pulley system is shown in figure 23. In this device there are two pulley wheels and one rope that is wound around them and affixed to the ceiling. A weight hangs from the second pulley wheel. Modelling this rope in SBF turns out to be tricky, and so, following Hegarty, it is divided into sections treated as separate "components" of a sort: rope A, hanging from the first pulley wheel, rope B that goes down from the first pulley wheel to the second, and rope C that goes up from the second pulley wheel to the ceiling. There are also two fixed ropes: that by which the first pulley hangs from the ceiling, and that by which the weight hangs from the second pulley. Each of these components is depicted by the expected basic shape in the drawing, and the structural

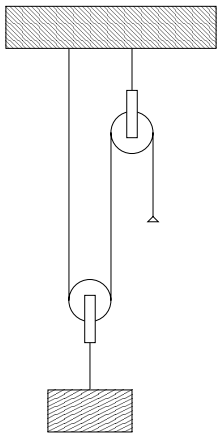


Figure 23: An example of a pulley system adapted from Schwartz and Hegarty [67]

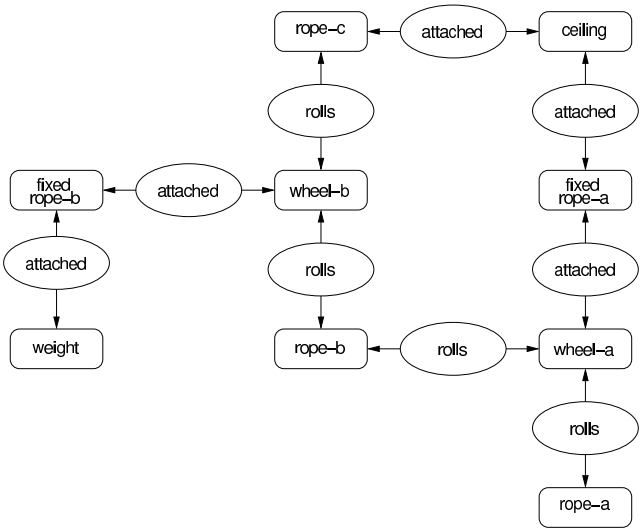


Figure 24: The structural model for the pulley example

relations between them (all binary, as always) are depicted by the expected composite shapes. The structural model is diagrammed in figure 24, the breakdown into basic and composite shapes is not shown.

3.3 Behavior and Function

The functional decomposition in Kritik's models, discussed in §3.1 above, began with a functional specification for the whole device, which was realized in a detailed behavior specified in terms of qualitative states and transitions. Those transitions could then be further decomposed, bottoming out in primitive functions. The primitive functions were functions of some component that achieved some basic operation, such as the allowing of a substance to pass through some component, or

Table 7: The schema for a behavior in SBF

Slot	Type	Number
Component or Quantity	*component* or *quantity*	1
States	*state*	1+
Transitions	*transition*	0+
Initial State	*state*	1
Initial Transition	*transition*	0–1
Final State	*state*	1

the creation or destroying of some substance. All this was based on the component substance ontology of early qualitative physics work and the ontology of functions developed by Bylander and Chandrasekaran [10, 9].

In this work I do not make extensive use of this function ontology, but I do preserve it. As such, components have primitive functions, which will be discussed below, and behavioral transitions can link to these primitive functions. Behaviors themselves are qualitative states either of components or quantities: *a* behavior is a behavior of either a *single* component or of a *single* quantity. A functional specification of a device then abstracts some particular behavior to represent the device's function.

3.3.1 Behavioral Models

When the entities of a device interact to produce behavior, the things doing the behaving are components and quantities. As such a behavior is of either a component or a quantity, and thus there are component behaviors and quantity behaviors. The variable parameters of these components or quantities are what change across the states and state transitions of these behaviors. The schema for a behavior is shown in table 7.

Behaviors are sequences of states and transitions. In the past, behaviors have always been linear sequences of states or else cyclic behaviors where the last state repeated the first. As such, each state had a unique successor and predecessor. In theory, there is no reason why branching behaviors, for example, are not possible, with multiple successor states to a single state, which would transform these behaviors into general finite state machines. But as this has not been necessary before, I preserve the old convention of the linear sequence.

A state, then, is a state *of* a component or quantity, and has a previous state and transition (the

Table 8: The schema for a behavioral state

Slot	Type	Number
Component or Quantity	*component* or *quantity*	1
Previous State	*state*	0–1
Previous Transition	*transition*	0–1
Next Transition	*transition*	0–1
Next State	*state*	0–1
Parameter Values	*expression*	0+

transition leading into it, and the state *from* which it leads), as well as a next state and transition (the transition leading out of it, and the state *to* which it leads). Most importantly, a state sets values to the parameters of the component or quantity of which it is the state. This schema is shown in table 8

Transitions are the core of behaviors in an important sense: the slots of transitions are what capture the causal relationships of behaviors. The transition itself simply represents succession in time, but transitions occur for some set of reasons, and there are a wide variety of possible reasons that can be represented in SBF. A transition of course connects two states, a previous and a next state. It can occur *as per* some physical principle, and *under the condition* of some other state or transition, or else a component, a structural relation, or even a quantity. Transitions can be *due to* an external stimulus, they can be further specified *by another behavior*, and finally transitions can occur *using a function* and specifically a primitive function of some relevant component. In addition to this, transitions can specify *parametric equations* that relate the parameter values in the previous and next states to each other (e.g. that heat increases from state 1 to state 2). All this is shown in the schema in table 9.

3.3.2 Functional Models

SBF employs two kinds of functions in the functional model: functional specifications as abstractions of behaviors, and primitive functions. It is this pair that allows for the representation of a functional decomposition of a device. A functional specification abstracts a particular behavior into (up to) two states: a *given* state and a *makes* state, and references the behavior that implements this function. Since behaviors can make use of external stimuli, this is also specified in the function. And finally, relationships between state are occasionally important, such as the heat increasing from

Table 9: The schema for a behavioral state transition

Slot	Type	Number
Previous State	*state*	1
Next State	*state*	1
As Per Principle	*physical principle*	0+
Under Condition State	*state*	0+
Under Condition Transition	*transition*	0+
Under Condition Component	*component*	0+
Under Condition Quantity	*quantity*	0+
Under Condition Structure	*structural relation*	0+
Using Function	*primitive function*	0+
By Behavior	*behavior*	0–1
Due to Stimulus	*external stimulus*	0–1
Parametric Equation	*expression*	0+

Table 10: The schema for a functional specification of a device

Slot	Type	Number
Device Model	*device model*	1
Given	*state*	0–1
Makes	*state*	0–1
By Behavior	*behavior*	0–1
Due to Stimulus	*external stimulus*	0–1
Provided	*expression*	0+

the given to the makes state, and so functions have a *provided* slot that serves the same role as the parametric equations of a transition. This schema is shown in table 10.

Primitive functions cannot be further decomposed into behaviors. SBF in the past has used an ontology of primitive functions inherited from previous work, including *allow*, *create*, *destroy*, and *pump*. As their names suggest, *allow* specifies that the function of a component is to allow some given substance (quantity) to pass through it, *create* and *destroy* that the component creates or destroys that substance, respectively, and finally *pump* that the component in question causes the movement of that substance. In the application of kinematics devices the only primitive function needed will be *allow*. For instance, since a revolute joint might allow either of the two components to move in general, the primitive function specifies that only one of them actually has that function. The schema for a primitive function, which is linked to the particular component, is shown in

Table 11: The representation of a primitive function of an SBF component

Slot	Type	Number
Function	*symbol*	1
Component	*component*	1
Quantity	*quantity*	1

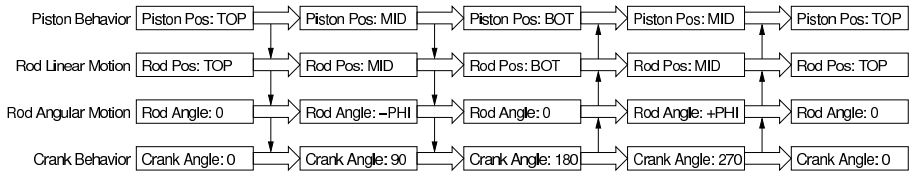


Figure 25: A diagram of the behavioral model of the piston and crankshaft device

table 11.

3.3.3 Piston and Crankshaft Example

In the piston and crankshaft example, each of the four quantities (linear motion of the piston, linear and angular motion of the connecting rod, and the angular motion of the crankshaft) has its own behavior, as shown in figure 25 (an example state schema is shown in table 12, and a transition in table 13). Here, the horizontal arrows indicate transitions, and the vertical arrows indicate *under condition transition* links between them. There is an external stimulus on the piston in the piston motion behavior in the first half of its motion. The crankshaft continues its motion *as per* the *principle* of rotational intertia due to its mass, and slows down *as per* the *principle* of friction, and

Table 12: An example behavioral state from the piston motion behavior

Slot	Value
id	p-s1
Quantity	Piston-Motion
Previous State	p-s2
Previous Transition	t-ps2-ps1
Next Transition	t-ps1-ps1
Next State	p-s2
Parameter Values	piston-position: cylinder top
	piston-direction: downward
	piston-velocity: *v*

Table 13: An example behavioral state transition from the piston motion behavior

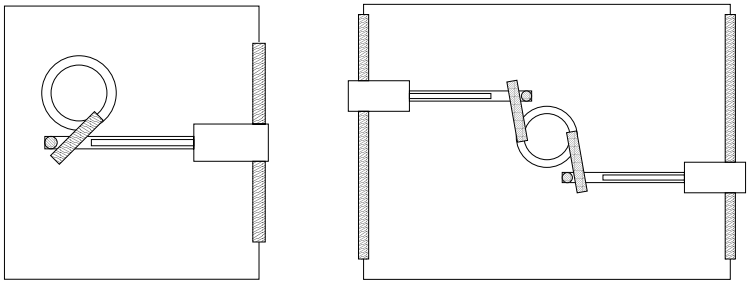
Slot	Value
Previous State	p-s1
Next State	p-s2
As Per Principle	—
Under Condition State	—
Under Condition Transition	—
Under Condition Component	—
Under Condition Quantity	—
Under Condition Structure	cylindrical-joint(cylinder, piston) revolute-joint(piston, connecting-rod)
Using Function	allow(piston, piston-motion)
By Behavior	—
Due to Stimulus	piston downforce
Parametric Equation	—

Table 14: The function of the piston and crankshaft example

Slot	Value
Device Model	piston-crankshaft
Given	c-s1
Makes	c-s2
By Behavior	crankshaft-motion
Due to Stimulus	—
Provided	—

all of the transitions occur *using* the relevant *allow function* of the appropriate component. The function of the device is to turn the crankshaft, and so the functional specification refers to the crankshaft behavior: the *given* state is the initial state of that behavior, and the *makes* state is one of the subsequent states (this functional specification is shown in table 14

There are several example targets associated with this source model, but the three that will be discussed in detail are those shown in figures 1(b), 1(c), and 1(d) on page 2. The first of these shows the same device in a different configuration: with the piston at the top of its range of motion. The other two show variations on this model with different *numbers* of the same components.



(a) A sample target drawing for the door latch example, with the door latch now pulled inward

(b) Another door latch target, this time a double door latch with one cam pulling two latches on either side of a door

Figure 26: Two sample door latch targets

3.3.4 Door Latch Example

The behavioral model is straightforward: there are three components that have behaviors (the cam, the shaft, and the bolt), and each has only two states. The cam has an initial state as shown in the figure, and a final state of being turned by some angle clockwise. This is caused by an external stimulus and occurs *under the condition* of the structural relation between the cam and the door. This causes (*under condition transition*) the shaft to go from its initial state of being in the forward position to being pulled back, which transition happens *under the condition* of the structural relation between the cam and the shaft. Finally, this causes (once again *under condition transition*) the bolt to be pulled back, again *under the condition* of the structural relation between the bolt and shaft. The function of the device is to pull back the shaft, and so the *given* and *makes* states in the function are the two states of the bolt's behavior.

Figure 26 shows two target drawings for this source model. The first shows the same device in the second state, with the bolt pulled back. The second is more complex, and involves one cam operating two bolts on either side of a door.

3.3.5 Pulley Example

The behavior of the two pulley system is easier to describe than it is to represent in a model. An external stimulus, in the form of a force on the rope (at the far right of the drawing), causes a downward motion on rope A, which turns the first wheel clockwise, which causes an upward motion

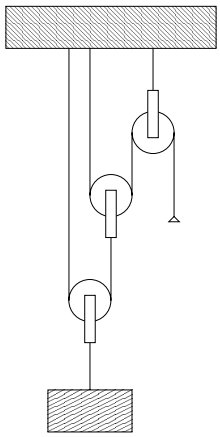


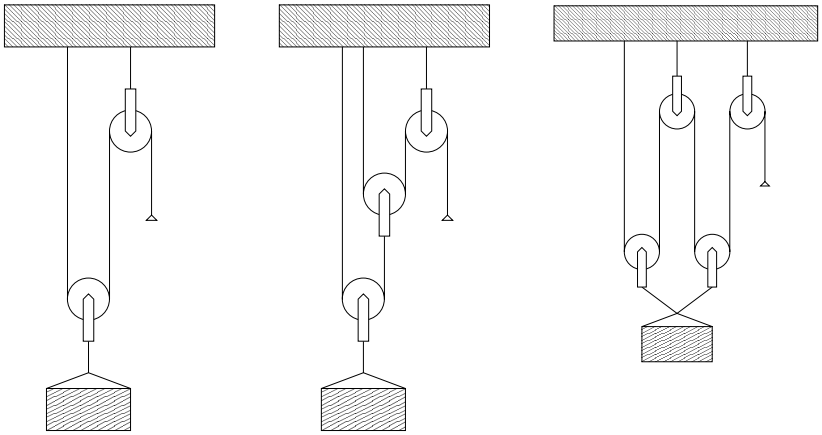
Figure 27: A target drawing of a pulley system adapted from Schwartz and Hegarty [67]

on rope B, which causes the second wheel to turn counterclockwise. This, in turn, causes that same wheel to raise, pulling up on its fixed rope and thus the weight. Each of these motions is a separate quantity with its own behavior, and all of the behaviors are two-state: still and moving. The function of the device is to raise the weight, and so the *given* state is the weight at a standstill (weight motion with velocity zero), and the *makes* state is the weight moving upwards, and this is realized *by* the *weight motion behavior*. That's nine components (two wheels, ropes A, B, and C, two fixed ropes, the ceiling, and the weight), and nine connections. Six of the components have behaviors (the two wheels, ropes A and B, the second fixed rope, and the weight), and one component (the second wheel) has both a linear and a rotational motion, thus there are seven behaviors.

Figure 27 shows a target drawing for this device, one of a more complex pulley system using three pulleys. Two more source drawings and four more target drawings will be considered when we come to the experiments. We now turn to the algorithms implemented in Archytas.

3.4 Additional Examples

As will be discussed in chapter 6, the original pulley example (figure 23) produced many problems, and in particular there were many redundancies in the shapes that caused far more matches than there should have been which were not caught in disconnected chain removal. Hence, a modified source and target drawing was attempted that slightly altered the shapes of the wheels and also the weight, shown in figure 28. This presented an improvement but there were still problems, and so a final pulley example was constructed, shown in figure 29. Here I present the drawings without



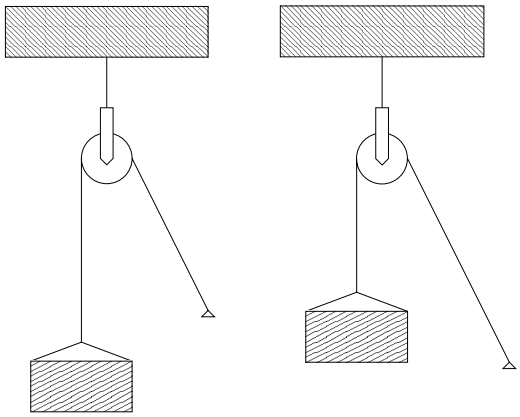
(a) A revised pulley example with a new source drawing

(b) A revised target for this example

(c) A second target drawing

Figure 28: A second set of pulley examples designed to eliminate some of the confusing redundancy of certain shapes

models. The source model of the pulley in figure 28(a) is identical to that of the original pulley example above. That of figure 29(a) is simpler, involving only a single pulley.



(a) The source drawing

(b) The target

Figure 29: A third pulley example illustrating a change in device state with no redundancy in component shapes.

CHAPTER IV

FROM DRAWING TO STRUCTURE

The goal of the shape mapping stage is to align shapes in the source and target so as to facilitate teh transfer of the structural model elements. However, at the start there are no shapes in the target, only a sea of individual lines and intersections, and so first Archytas attempts to align the individual shape patterns from the source with the augmented line intersection graph of the target. In particular, Archytas attempts to find mappings from each basic and composite shape in the source to some subgraph of the target's representation. Here, the mappings must be exact: each element of a shape pattern from the source must match, and so the algorithm is computing *subgraph isomorphism*.[1]

Since we have both basic and composite shapes, the method must operate at several levels of aggregation. First, it must find the low-level patterns of lines and arcs and intersection points that make up individual shape patterns. Then, these being found, it must find patterns of whole shapes that might depict structures in the device depicted in the target drawing, establishing a mapping from source shapes (whole shapes, now, not points and lines any longer) to target shapes. This being found, in the next chapter I will discuss methods for transferring structure, behavior, and function from source to target on the basis of the shape-level mapping.

The input, then, to this shape analogy method is a target drawing represented as an augmented line intersection graph, and a set of basic and composite source shapes represented as subgraphs of the source drawing's augmented line intersection graph. The problem is finding the shapes in the target and establishing mappings from the source to the newly transferred target shapes. The first step is the matching of shape patterns at the level of point and line and arc segment, and here I shall begin.

The shape representation devised in chapter 2 has been devised with this goal in mind: that a structured matching of a shape pattern to a drawing should match in just those places where

[1] Algorithms for subgraph isomorphism are sometimes referred to as "graph matching" algorithms, a natural but potentially quite confusing term which I will avoid, as "graph matching" has an entirely separate meaning in the graph algorithm literature.

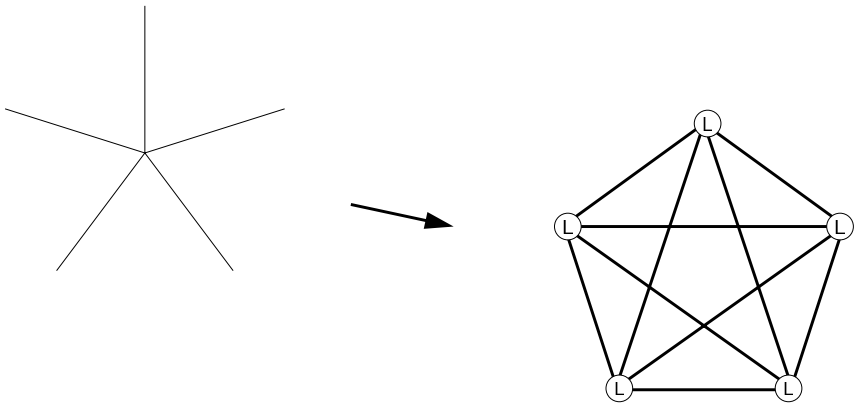


Figure 30: A simple example of a shape who's line intersection graph is non-planar

that shape can and should be found. As such, we need a method of structural pattern recognition. Structural pattern recognition as alignment of symbol structures (subgraph isomorphism, in graph-theoretic terms) is an old idea in computer science; Suetens et al. [76, §3] survey its use in object recognition, for instance, and Ullman [80] devised a well known algorithm. In general, subgraph isomorphism is NP-Complete, but David Eppstein [22] gives reason to hope with his proof that, for a fixed pattern using only *planar* graphs, subgraph isomorphism can be done in *linear* time in the size of the "text" graph (here, the target drawing representation).

Although of course the drawings themselves are planar graphs, my line intersection graphs are, in point of fact, not always planar. Figure 30 shows an example of a shape who's intersection graph is the complete graph on five vertices, K_5. In general, when k lines meet at a point, or all intersect each other (e.g. a k-pointed star), one will get a k-clique in the line intersection graph. Such shapes are somewhat pathological, however, and in general we might expect the representations of most shapes to be more well-behaved, and so approximately planar if not actually planar. For instance, a n-sided polygon will generate an n-cycle in the line intersection graph, and each non-intersecting chord (that is, a chord that doesn't intersect with any other chords) will add a single vertex and two edges to it. It may be possible to use this information to gain some sort of speedup over a brute-force algorithm in this case, though this remains an open empirical question.

Following earlier work [86], I transform the problem of structured matching into a constraint satisfaction problem. In this application the constraints are purely spatial, but the technique allows us to employ such constraints as can be applied to the problem. A constraint satisfaction problem

takes a set of *variables* with potential *values* from one or more *domains* and attempts to find an *assignment* of values to variables under some *constraints*, where the constraints serve to characterize which assignments of values to variables are allowed. It is well known that the structure of the network of constraints over the variables can determine the complexity of a constraint satisfaction problem [33, 34, 45]. In this work, I employ a basic backtracking algorithm, where the problem now is not to find a *single* assignment of values to variables but to find *all* assigments: if a given shape occurs several times in a drawing it's important to find out about all of these occurrences.

Stated another way, the problem in general is NP-Complete, the known algorithms are all exponential time worst-case complexity, and the algorithms I present below are not exceptions (if they were I would have solved the P vs. NP problem). However, constraint satisfaction algorithms are to some extent *data-dependent*, meaning their *expected* or *average* running time in general may be less than exponential if the data exhibit certain kinds of structures. That is to say, if there are n variables and m values in a constraint satisfaction problem (here, the variables are the source shape pattern's intersection points and the values are all of the intersection points across the whole target drawing), then the *worst-case* complexity for any backtracking method is $O(n^m)$, and if there are k shapes to test for (where, to simplify somewhat, each shape has n intersection points), the final algorithm will be $O(kn^m)$ as the search must be repeated for each shape. More sophisticated algorithms such as back-jumping algorithms and forward checking methods and intelligent variable ordering are required to take full advantage of the data-dependent nature of constraint satisfaction problems, but as they are all variations and optimizations of the basic backtracking algorithm, I employ here the basic backtracking algorithm, and leave such optimizations to future work.

In order to match the shapes in the source drawing to the target, Archytas first begins by trying to pattern match each composite shape to the target line intersection graph. Upon finding the occurrences of these shapes, Archytas breaks the composite shape mappings into corresponding basic shape mappings, so that when two composites with a basic shape in common overlap in the mapping as well, that basic shape can be inferred of the target.

Stated more formally, let's say a composite source shape S_X is made up of two basic shapes S_A and S_B, so that $S_X = \mathfrak{L}(G)[V_{S_A} \cup V_{S_B}]$, and another source composite shape S_Y is made up of two others, $S_Y = \mathfrak{L}(G)[V_{S_B} \cup V_{S_C}]$. Thus, the overlap between the two composites is just the basic shape

they have in common: $S_X \cap S_Y = S_B$. Then, when we have two mappings m_{S_X} and m_{S_Y}, we can look for an overlap between them, $m_{S_X} \cap m_{S_Y}$, and when we do, we should expect it to be a valid mapping of the basic shape S_B, that is $m_{S_X} \cap m_{S_Y} = m_{S_B}$. This may not be true, but when it is, we can infer by analogy that the basic shapes S_A, S_B, and S_C as well as the composites S_X and S_Y are present in the target. Furthermore, we can return a shape-level mapping with assignments of each of these source shapes to newly transferred target shapes based on these mappings.

In what follows, first I will go through the reduction of subgraph isomorphism in augmented line intersection graphs to a constraint satisfaction problem, and then briefly describe the backtracking algorithm. Then, I will discuss the issues involved in moving from individual mappings of particular shapes at the level of intersection point and line and arc segment to mappings of whole shapes. In particular, I will discuss the logic behind the method in which all the mappings of a particular shape are grouped into sets of symmetric mappings so that one shape is transferred *per group* rather than per mapping. The presentation of this transfer method will conclude the chapter.

4.1 Shape Matching as Constraint Satisfaction

4.1.1 Constraint Satisfaction

For any set D, and any natural number n, the set of all n-tuples (ordered lists of the form $\langle x_1, \ldots, x_n \rangle$) over D is written D^n. For any tuple $t \in D^n$, and any i in the range 1 to n, the value of the ith coordinate position of t is denoted by $t[i]$, so the whole tuple might be written $\langle t[1], t[2], \ldots, t[n] \rangle$. A n-ary *relation* R over D is any subset of D^n.

A *constraint satisfaction problem* (CSP) of n variables consists of:

- a finite set of variables V, where $n = |V|$
- a finite domain of values D
- a set of constraints $\{C_1, C_2, \ldots, C_k\}$

where each constraint has a *scope* (the set of variables to which it applies) and a constraint relation (the values from the domain it allows for those variables). Specifically, $C_i = (S_i, R_i)$, where the scope S_i is an m_i-tuple of variables, $S_i \in V^{m_i}$, and the constraint relation R_i is an m_i-ary relation over the domain D. The number m_i is known as the *arity* of the constraint. The constraint applies the

values in the relation $R_i \subseteq D^{m_i}$ to the variables in S_i, admitting these combinations of values and no others to those variables.

In this application, m_i will always be 2, as we shall see, so that all the constraints are binary. Usually, the decision problem for a constraint satisfaction problem is determining if there is *some* set of values to assign to each variable that satisfies all of the constraints (and usually also of giving some such assignment). In particular, we shall take solutions as mappings of the form $m : V \to D$, so that for some variable x_i, $m(x_i)$ is the domain value assigned to that particular variable. If a constraint C has the scope (x_i, x_j), and R is the constraint relation, then if m is a valid solution to the constraint satisfaction problem, we know that

$$\langle m(x_i), m(x_j) \rangle \in R$$

and this is true for all of the constraints C. Again, in this application we are not interested simply in the decision problem, but rather in the problem of finding *all* the valid assignments.

4.1.2 Subgraph Isomorphism in Augmented Line Intersection Graphs

Let $G = (V, E)$ and $G' = (V', E')$ be any two undirected[2] graphs, and let $L : E \to \Lambda$ be a labelling of the edges of G and $L' : E' \to \Lambda'$ be a labelling of the edges of G'. Note that these labels are not expected to be unique within each graph—that is, two edges may share a label (otherwise defining the labelling function would be superfluous from a mathematical perspective). G and G' are said to be *isomorphic*, written $G \simeq G'$, if there exists a bijective mapping $m : V \to V'$ such that:

- $e = \{x, y\} \in E$ if and only if $e' = \{m(x), m(y)\} \in E'$ for all $x, y \in V$
- For all e, f, with $e = \{x, y\}$ and $f = \{z, w\}$, $m(x) = x'$, $m(y) = y'$, $m(z) = z'$, $m(w) = w'$ with $e' = \{x', y\} \in E'$ and $f' = \{z', w'\} \in E'$, then $L(e) = L(f)$ if and only if $L'(e') = L'(f')$

This function m is called an *isomorphism*. If there is a subgraph $H' \subseteq G'$ such that $G \simeq H'$, with $m : V \to V_{H'}$, then we say that m is a *subgraph isomorphism*, that is an isomorphism of G onto a *subgraph* of G'. In general, graph and subgraph isomorphism need not be formulated in terms of a

[2]Directed graphs use a somewhat simpler notation than undirected graphs, so that an edge $e \in E$ is written $e = uv$, $u, v \in V$. In undirected graphs, the edges are unordered pairs, i.e. sets of two vertices, $e = \{u, v\} \subseteq V$.

labelling, but it is a common way to formulate the problem (one that casts and eye towards the most common applications of graph isomorphism algorithms), and we will find it useful here.

Recall the definition of a line intersection graph from §2.2.2.2 (p.29). Connecting the notation of line intersection graphs to the above definition, if the graph P is a source drawing regarded as a plane graph (as per the discussion in chapter 2), and if $\mathfrak{L}(P)$ is the line intersection graph, then let $G = S \subseteq \mathfrak{L}(P)$, where S is a shape from the source image; if Q is the target image then $G' = \mathfrak{L}(Q)$; finally, let L and L' map an edge e_{x,ℓ_1,ℓ_2} to the point x at which the line/arc sets ℓ_1 and ℓ_2 intersect. If there is more than one intersection, then there will be more than one edge between ℓ_1 and ℓ_2, with each edge labelled by one of the intersection points. Once again, note that it is possible for several lines to all intersect at the same point, so that there will be a separate edge for each pair, each edge with the same label. Then we are searching for a subgraph $S' \subseteq G = \mathfrak{L}(Q)$ such that $S \simeq S'$. In this work once again we would like to enumerate *all* such subgraphs S'.

For any graph isomorphism $m : V_G \to V_H$, for any two graphs G and H, it is possible to define an *edge mapping* $m_E : E_G \to E_H$, so that, if $m(u) = x$ and $m(v) = y$ and $e = \{u, v\} \in E_G$ then $m_E(e) = f$, where $f = \{x, y\} \in E_H$. Likewise, in line intersection graphs, for a given (subgraph) isomorphism $m : S \to S'$ from a source shape to a (newly discovered) target shape, we can also define the edge mapping m_E in the same fashion. Specifically, if

$$m(\ell_1) = \ell_a$$

$$m(\ell_2) = \ell_b$$

and there is an edge $e_{x,\ell_1,\ell_2} \in E_S$ and an edge $e_{y,\ell_a,\ell_b} \in E_{S'}$, then

$$m_E(e_{x,\ell_1,\ell_2}) = e_{y,\ell_a,\ell_b}$$

Note that, in general, there may be *two* edges (intersections) between any given pair of line sets ℓ_1 and ℓ_2 if one of them is an arc.[3] If this is the case, *both* source intersections must have target analogs for the mapping to be consistent. In terms of notation, I will write m_V for the vertex mapping and m_E for the edge mapping when it is important to distinguish them, and otherwise speak of a mapping m as mapping *both* vertices and edges.

[3]Note that, due to the full breakdown done during pre-processing of the drawing discussed in chapter 2, anything still labelled as a "circle" in Archytas must have at most one intersection, or else it would have been divided into arc segments.

4.1.3 Subgraph Isomorphism as a Constraint Satisfaction Problem

The above definition of subgraph isomorphism clearly suggests that the variables of the constraint satisfaction problem are the vertices (which are the line and arc sets) of the augmented line intersection graph of the source shape, with the consistency of the intersection points forming the constraints. However, for historical reasons,[4] Archytas treats the primary variables of the constraint satisfaction problem as the *intersection points*, i.e. the *edges* of the graph, using the mapping of line and arc sets (the vertices) to form the constraints between variables. This proved to have some limitations that required a clumsy workaround, which will be discussed later in the chapter. The more parsimonious reduction is of course the obvious one just stated, and in future work that is the approach I plan to take.

In subgraph isomorphism one sometimes speaks of the "text" graph versus the "pattern," where the goal is to structurally align the pattern with the text (e.g, Eppstein [22] speaks this way). In this work, the target image is the "text" and the shape is the "pattern." Following the above discussion of subgraph isomorphism, I will first reduce the problem to constraint satisfaction in general graph theoretic terms, and then return to the notation of line intersection graphs.

Once again, given a pattern G and a text graph G', the subgraph isomorphism problem is one of finding a subgraph $H' \subseteq G'$ such that $G \simeq H'$. Reducing this to a CSP, I take the variables to be the *edges* E_G of the pattern, and the domain therefore to be the edges $E_{G'}$ of the text. It remains to define the constraints. For each pair of edges $e, f \in E_G$, when these edges share an endpoint $v \in V_G$, then for any mapping $m : E_G \rightarrow E_{G'}$, $m_E(e) = e'$, $m_E(f) = f'$ if and only if e' and f' are both incident on some vertex $v' \in V_{G'}$, with $m_V(v) = v'$, and furthermore, $L(e) = L(f)$ if and only if $L'(e') = L'(f')$. More formally, for each pair of incident edges $e, f \in E_G$ we define two constraints, C_i,C_j, with $S_i = (e, f)$ and $S_j = (f, e)$, so that their relations R_i and R_j contain all pairs of edges from the text G' that are incident on each other, and furthermore if the pattern edges have the same label then the pairs in the relation all have the same label and likewise if the pattern edges have different labels then the pairs in the relation all have different labels.

[4] In particular, the first version of Archytas [85], not discussed in this dissertation, employed a variation of a maximum common subgraph algorithm on whole drawings, whereby the number of edges mapped were maximized rather than the number of vertices (following Chen and Yun [14]). This method was reused and simplified to a subgraph isomorphism method in the version of Archytas described here.

In addition to this, there is an interaction between the edge mapping and the vertex mapping which must be spelled out. For each edge assignment $m_E(e) = e'$, if $e = \{u, v\} \in E_G$ and $e' = \{x, y\} \in E_{G'}$ then that edge map implies one or the other of a pair of vertex mappings:

$$m_V(u) = x \qquad\qquad m_V(u) = y$$
$$m_V(v) = y \qquad\qquad m_V(v) = x$$

For a particular mapping, only one of these, either the one on the left or the one on the right, can be chosen. As such, if $e = \{u, v\}$ and $f = \{v, w\}$, and if $m_E(e) = e'$, $m_E(f) = f'$, where $e' = \{x, y\}$ and $f' = \{y, z\}$, then

$$m_V(u) = x$$
$$m_V(v) = y$$
$$m_V(w) = z$$

and there is no other vertex assignment consistent with this edge assignment. If it were the case that $f' = \{x, z\}$, then it would be necessary for $m_V(u) = y$, $m_V(v) = x$ and $m_V(w) = z$.

In graph theory one sometimes speaks of *edge units*, which is an edge considered together with each of its endpoints. Here we see that the variables of the constraint satisfaction problem are in fact edge units in this sense, and because of that what appears to be a single assignment $m(e) = e'$ is actually two different assignments once the edges' endpoints are taken into consideration, and the constraints must then capture this. Thus, for some constraint C_i with scope $e, f \in E_G$, the constraint relation R_i is not over mere edge assignments, but rather over edge *unit* assignments, and must allow only those edge *unit* assignments that are consistent.

Connecting all this back to line intersection graphs once again, we again take the graph P to be a source drawing regarded as a plane graph and with $\mathfrak{L}(P)$ being the line intersection graph, then we let $G = S \subseteq \mathfrak{L}(P)$, where S is a shape from the source image. We take Q to be the target image, then, so that again $G' = \mathfrak{L}(Q)$. And finally, once again, we take L and L' to be labellings that map an edge e_{x,ℓ_1,ℓ_2} to the point x at which the line/arc sets ℓ_1 and ℓ_2 intersect. And again, if there is more than one intersection, then there will be more than one edge between ℓ_1 and ℓ_2, with each edge labelled by one of the intersection points. Once again, note that it is possible for several lines to all intersect

at the same point, so that there will be a separate edge for each pair, each edge with the same label. We are then using constraint satisfaction to enumerate all the subgraphs $S' \subseteq G'$ such that $G \simeq S'$. The constraints are defined exactly as above, though below we will constrain things still further.

4.1.4 Backtracking Constraint Satisfaction in Line Intersection Graphs

One of the most common methods for solving constraint satisfaction problems is backtracking, along with all its many variants [7, 58, 55]. It is a simple method that has been rediscovered many times [7]. The algorithm is best stated recursively. It takes an existing partial assignment of values to some of the variables in the problem, and then extends it with an assignment to the next variable. If this assignment is consistent with all the previous assignments, then recurse and continue, otherwise discard it. If the goal is to produce just *one* valid assignment, then on the recursion the method returns whatever the recursive step does, and on failure it tries the next value, returning failure itself if all the values have failed. If the goal is to find *all* valid assignments, then the method continues to try all the values whatever the result of the recursive call—this is the only difference between returning one and returning all solutions. The bottom of the recursion is when the method recurses but there are no more variables to assign, then in this case it records the assignment and returns. This algorithm is stated more precisely in table 15. On the first call, m and M are both empty, and $k = 1$. It remains to consider the consistency test.

The constraint satisfaction problems discussed here are all *binary* constraint satisfaction, and since all the constraints are binary the consistency test in the backtracking algorithm need only consider pairs of variables. In particular, since it's being handed a variable, it only needs to iterate over single variables (rather than, say, a double iteration over pairs of variables, which would be necessary for ternary constraints). The algorithm is shown in table 16. For a perfectly complete algorithm, we would also have to iterate over constraints with the scope (V_k, V_i), but in this application all the constraints are symmetric[5] so that step is unnecessary.

In applying this algorithm to the problems considered here, there is only one further consideration that needs to be stated. The backtracking algorithm simply iterates over all values $x \in D$, but in this case these values are edges $e \in E_{\mathfrak{L}(Q)}$, the intersection points in the target drawing, and

[5]That is, in any graph if an edge e is adjacent to an edge f then f is of course adjacent to e, and so the constraint that they be adjacent and either share a label or not is not sensitive to the order of the variables.

Table 15: The basic backtracking algorithm for constraint satisfaction

backtrack (m,k,V,D,C,M)

Input:	current partial assignment $m : V \to D$ an index $k \in \{1,\ldots,n\}$, $n = \|V\|$ set of variables $V = \{V_1,\ldots,V_n\}$ set of domain values D set of constraints C all valid assignments so far M
Output:	all valid assignments M

```
1: if k > n then
2:     return M ∪ {m}
3: else
4:     M' ← M
5:     for all x ∈ D do
6:         if consistent(V,k,x,m,C) then
7:             M' ← backtrack(m ∪ {V_k = x}, k+1, V, D, C, M')
8:     return M'
```

Table 16: The consistenct test for the backtracking algorithm in table 15

consistent (V,k,x,m,C)

Input:	set of variables $V = \{V_1,\ldots,V_n\}$ an index $k \in \{1,\ldots,n\}$, $n = \|V\|$ the value x to try for variable V_k current partial mapping $m : V \to D$ set of constraints C
Output:	*true* or *false*

```
1: if k > 1 then
2:     for i ∈ {1,...,k−1} do
3:         for all constraints C_i ∈ C with S_i = (V_i, V_k) do
4:             if m(V_i) = x or ⟨m(V_i), x⟩ ∉ R_i then
5:                 return false
6: return true
```

from above we know that in fact there are *two* assignments of the form $m(V_k) = x$. Thus, the central recursive step actually must try both of these in turn, each of which may result in a separate mapping.

4.1.4.1 Augmenting the Constraints

As stated, the algorithm in tables 15 and 16 will calculate only the simplest structural alignments; squares will map to rhombuses and trapezoids, for instance. There are several additional pieces of information available in the *augmented* line intersection graphs that can be used for constraints. For the intersection points themselves there are two additional constraints:

1. If the source intersection point (the pattern, the variable in the CSP) is one between perpendicular lines, then so must the target, otherwise the target's perpendicularity is irrelevant.

2. If two source intersection points border a common face or cell in the source drawing then so must the targets to which they are mapped, otherwise if they do not, the targets must not either.

The first of these is a singulary constraint rather than a binary one, and so can be tested when the next value is chosen from the domain, before the consistency test is called. The second one is a binary constraint *not necessarily between adjacent edges*.

Each edge unit assignment implies a pair of vertex assignments, and vertex assignments must also be consistent with the vertex assignments of other edge unit assignments. Hence, an additional set of constraints. The constraints for vertex (i.e. line or arc set or circle) assignments are as follows:

1. If the source vertex is a line set, so must be the target; if it is an arc set, so must be the target; if it is a circle, so must be the target

2. If two source line sets are disconnected but collinear, then so must be their targets, otherwise if the sources are not then the targets must not be either

3. If two source arc sets are disconnected but co-circular, then so must be their targets, and vice versa

The consistency of vertex assignments is tested when two edge assignments are compared. Since one edge assignment implies two vertex assignments, then the final constraint on consistent edges is that the two-by-two comparison of their implied vertex assignments shows that they are all consistent. This, then, defines the complete set of constraints for the constraint satisfaction problem of subgraph isomorphism in augmented line intersection graphs.

4.1.4.2 Completing the Mapping

Because the constraint satisfaction technique described here begins with *intersection points* as the variables over which the algorithm iterates, it will fail to map any individual lines or arcs or circles that do not intersect any other lines in the drawing. As an example of this, the piston and crankshaft drawings all contain circles that do not intersect other lines (e.g. the connecting rod has a circle at either end depicting the bearing between it and the piston as well as between it and the crankshaft). These disconnected circles show up as isolated vertices in the line intersection graph. Dealing with them is not difficult, however: we simply take each resulting mapping and "complete it" to include any disconnected pieces. This step is klunky and should be unnecessary but it is a consequence of how the algorithm has been designed.

As it turns out, these disconnected pieces are not as disconnected as they seem. In fact, without fail they are floating within some face in the drawing, so that they will be marked as being "contained" within this face in the graph dual. Recall that faces are linked to all the vertices and edges on their boundary, and this includes "inner" faces or edges as well as the outer boundary. So, the final remaining step is to search for all such disconnected lines, arcs, or circles that are contained within some face in the source shape. The boundaries of these faces should be mapped onto the target, which will give us a face in the target in which Archytas should find a target for the disconnected source line/arc/circle. Finding it, Archytas extends the mapping appropriately, and failing, Archytas discards the mapping as being incomplete.

4.2 Symmetric Mappings

In applying the above methods, a very interesting problem presents itself. For example, in the piston and crankshaft example, the composite shape for the piston and cylinder, an H-shape, will map onto an identical pattern in a target drawing *four* times. These four mappings are the four "symmetries"

of this shape. In mathematics, a *symmetry* of a shape is a mapping of that shape onto itself. For instance, a square can be rotated 90° about its center and the result will be indentical to the original. A *symmetry groups* characterizes a shape by the number of different ways it can be rotated, flipped, or even stretched (e.g. in the case of a line or certain kinds of repeating patterns) to produce the exact same figure. A square has eight symmetries, the H-shape of the piston and crankshaft (which can be seen in figure 17 (page 38) has four:

1. the identity mapping (i.e. no transformation)
2. a 180° rotation
3. a horizontal reflection
4. a vertical reflection

This observation has an important consequence: any subgraph isomorphism algorithm should be able to find *all* of these mappings for *each* occurrence of the figure in the drawing.

If we were to take just one mapping for each composite shape and attempt to find the overlap (i.e. the basic shape they have in common) then this fact would trip us up. For instance, take the horizontally flipped mapping of the piston/cylinder composite with the identity mapping of the piston/connecting-rod composite. If the piston's four corners are denoted by a, b, c, d and the target rectangle's by w, x, y, z then these two mappings will yield:

$$m_1(a) = x \qquad m_2(a) = y$$
$$m_1(b) = y \qquad m_2(b) = x$$
$$m_1(c) = z \qquad m_2(c) = w$$
$$m_1(d) = w \qquad m_2(d) = z$$

These are two *different* mappings, and so comparing them the overlap will *not* be found—unless we realize that the two mappings map the same *set* of points from the source shape onto the same *set* of points in the target. This is precisely the fact that makes two shapes symmetries what they are—namely, symmetries of the very same shape.

Table 17: The lookup method for finding the symmetric mapping set of a given mapping

findgroup (m, G)

Input:	a mapping m for some basic or composite shape the family G of sets of mappings for this shape
Output:	the mapping set G of the mapping m or nil

```
1: G' → the first mapping set in G
2: while G is nil do
3:    if Rng m = Rng m' for some m' ∈ G' then
4:       G ← G'
5:    else
6:       G' ← the next mapping set in G
7: return G
```

We can characterize this formally. Let Dom f be the *domain* and Rng f be the *range* of some mapping f. That is, if we define f as $f : A \rightarrow B$, then Dom $f = A$ and

$$\text{Rng}\, f = \{b \in B | \exists a \in A\ f(a) = b\}$$

The set B is called the *co-domain* of f, and in general may not be equal to the range. This will certainly be the case for any shape mappings in this application: the co-domain of most of these mappings is the entire target line intersection graph, whereas the range will be some subgraph of that, some figure within the drawing.

We shall say that two mappings m_1 and m_2 are *symmetric* if and only if

$$\text{Dom}\ m_1 = \text{Dom}\ m_2$$

and

$$\text{Rng}\ m_1 = \text{Rng}\ m_2$$

From this one very basic definition the solution to our problem becomes clear: two mappings from the same shape (basic or composite)—that is, with the same domain—can be grouped together into a mapping *set* when their ranges are equal. Thus, the two mappings of the piston shape shown above would be grouped together as two symmetries of the same target shape. The algorithm for looking up the symmetric mapping set of a given mapping is shown in table 17.

The shape mapping algorithm (which will be given in full at the end of this chapter) searches for mappings by composite shape and then divides these mappings into basic shapes. If G (with a slight shift in notation) is the source drawing's representation and $C \subseteq G$ is a composite shape, with $B_1, B_2 \subseteq C$ the two basic shapes that compose it, and G' is the target representation with $m : C \rightarrow G'$ a mapping for this shape, with m_{E_C} and m_{V_C} referring to the edge and vertex mappings respectively, then we can divide this into basic shape mappings:

$$m_{E_{B_1}} = m_{E_C}[E_{B_1}] = \{(x,y) \in m_{E_C} | x \in E_{B_1}\}$$
$$m_{V_{B_1}} = m_{V_C}[V_{B_1}] = \{(x,y) \in m_{V_C} | x \in V_{B_1}\}$$
$$m_{E_{B_2}} = m_{E_C}[E_{B_2}] = \{(x,y) \in m_{E_C} | x \in E_{B_2}\}$$
$$m_{V_{B_2}} = m_{V_C}[V_{B_2}] = \{(x,y) \in m_{V_C} | x \in V_{B_2}\}$$

Here, of course, $m[X]$ specifies a restriction of a mapping $m : A \rightarrow B$ to some subset $X \subseteq B$. Furthermore, if G_C is the set of symmetric mappings for the composite C to which m belongs, and the above basic shape mappings (call them m_{B_1} and m_{B_2}) belong to G_{B_1} and G_{B_2}, then we can link G_C to G_{B_1} and G_{B_2} so that, in transfer, the transferred composite is composed of the appropriate basic shapes.

There is only one complication with this this method: two different but symmetric composite shape mappings may nevertheless divide into basic shapes in *two different ways*—that is, the divided basic shape mappings may not be in the same symmetric mapping sets. To take a simple example, two rectangles that touch along one side will appear in the line inteersection graph as two 4-cycles (squares or diamonds when the graph is drawn) that share a single vertex. If each rectangle (each 4-cycle in the intersection graph) is a basic shape, then there are two ways to divide it. Let's call the basic shapes A and B in the source and the two adjacent 4-cycles X and Y in the target. Then if one mapping is the right flip or rotation of the other than they will be symmetric but one will map A to X and B to Y but the other will do just the opposite and map A to Y and B to X. Furthermore, these two divisions are incompatible: the drawing can only be regarded as representing things one way or the other but not both at the same time.

In order to deal with this, we must recognize that a given composite shape mapping set G_C may be divided into basic shape pairs in *more than one way*. For instance, if C_C contains 16 mappings

but *two* pairs of basic shape mapping sets, $G^1_{B_1}, G^1_{B_2}$ and $G^2_{B_1}, G^2_{B_2}$, then when the transfer step occurs we can treat this as *two* sets of composite shape mappings: G^1_C, G^2_C, with only one pair for each of these, and furthermore the resulting shape-level maps are marked as conflicting with each other (conflicting shape maps will be discussed below), so that one or the other will hopefully be removed later. The actual splitting of the one mapping set into multiple mapping sets does not occur until the transfer stage, described below; in the grouping algorithm Archytas merely links composite shape mapping sets with pairs of basic shape mapping sets, possibly more than one pair. This algorithm is shown in table 18.

4.3 Shape Mapping and Transfer

Now, finally, we are all set to state the overall shape transfer algorithm. The outline of the algorithm is as follows:

1. Apply each composite shape, matching it to the target drawing as many times as possible using backtracking constraint satisfaction with the composite shape elements as variables, target intersection graph elements as values, and matching graph structure as constraints.
    - Group symmetric composite shape mappings
2. For each composite shape mapping, break it into its basic shape mappings
    - Group symmetric basic shape mappings
3. For each set of symmetric basic and composite shape mappings, instantiate a new shape in the target drawing
4. Return a mapping from the source shapes to the target shapes that each one instantiated

This algorithm is described in detail in table 19.

At this level there is only one remaining complication: individual shape-level maps can conflict with each other. There are two ways in which this can happen. The first, described above, is that a given composite shape is split into basic shapes in two or more different ways. When this happens the resulting composite shapes as well as the resulting pairs of basic shapes are all marked as conflicting with each other. The second way is that occasionally a composite will map onto

Table 18: The algorithm for splitting and grouping composite shape mappings

group (m, C, B_1, B_2)

Input:	a mapping m from C to the target a source composite shape C source basic shapes $B_1, B_2 \subseteq C$
Output:	the mapping set G_C of the mapping m the mapping sets G_{B_1}, G_{B_2} of $m[B_1], m[B_2]$

1: Let G_C be the family of all mapping sets for the composite shape C found so far
2: Let G_{B_1} be the family of all mapping sets for the basic shape B_1 found so far
3: Let G_{B_2} be the family of all mapping sets for the basic shape B_2 found so far
4: $m_{B_1} \leftarrow m[B_1]$
5: $m_{B_2} \leftarrow m[B_2]$
6: $G_C \rightarrow \text{findgroup}(m, G_C)$
7: **if** $G_C \neq \text{nil}$ **then**
8: $\quad G_C \leftarrow G_C \cup \{m\}$
9: **else**
10: $\quad$ Let G_C be a new mapping set, with $G_C = \{m\}$
11: $\quad G_C \leftarrow G_C \cup \{G_C\}$
12: $G_{B_1} \rightarrow \text{findgroup}(m_{B_1}, G_{B_1})$
13: **if** $G_{B_1} \neq \text{nil}$ **then**
14: $\quad G_{B_1} \leftarrow G_{B_1} \cup \{m\}$
15: **else**
16: $\quad$ Let G_{B_1} be a new mapping set, with $G_{B_1} = \{m_{B_1}\}$
17: $\quad G_{B_1} \leftarrow G_{B_1} \cup \{G_{B_1}\}$
18: $G_{B_2} \rightarrow \text{findgroup}(m_{B_2}, G_{B_1})$
19: **if** $G_{B_2} \neq \text{nil}$ **then**
20: $\quad G_{B_2} \leftarrow G_{B_2} \cup \{m\}$
21: **else**
22: $\quad$ Let G_{B_2} be a new mapping set, with $G_{B_2} = \{m_{B_2}\}$
23: $\quad G_{B_2} \leftarrow G_{B_2} \cup \{G_{B_2}\}$
24: Link G_C to G_{B_1}, G_{B_2}
25: **return** G_C

Table 19: The shape transfer algorithm

shape-analogy (B, C, T)

Input:	source basic shapes B source composite shapes C target line intersection graph T
Output:	shape-level mapping M from source shapes to new target shapes

1: **for all** basic shape $B_i \in B$ **do**
2: Let $G_{B_i} = \varnothing$
3: **for all** composite shape $C_i \in C$ **do**
4: Let $G_{C_i} = \varnothing$
5: Let $M = \varnothing$
6: **for all** $C \in C$ **do**
7: Find all consistent mappings $m_C : C \rightarrow T$ using the backtracking algorithm
8: **for all** composite shape mappings $m_C, C \in C$ **do**
9: Let $B_i, B_j \in B$ be the basic shapes $B_i, B_j \subseteq C$
10: $G_C, G_{B_i}, G_{B_j} \leftarrow \text{group}(m, C, B_i, B_j)$
11: **for all** G_B for each basic shape B **do**
12: **for all** $G_B \in G_B$ **do**
13: Let m_B be any mapping in G_B
14: Let $E = \{e \in E(T) : \exists x \in E(S),\ m_{E_B}(x) = e\}$
15: Let $V = \{v \in V(T) : \exists y \in V(S),\ m_{V_B}(y) = v\}$
16: Let $T_B = (E, V)$ be a new target basic shape
17: Create a new *shape-level* map $M(B) \mapsto T_B$
18: **for all** G_C for each composite shape C **do**
19: **for all** $G_C \in G_C$ **do**
20: Let m_C be any mapping in G_C
21: Let $E = \{e \in E(T) : \exists x \in E(S),\ m_{E_C}(x) = e\}$
22: Let $V = \{v \in V(T) : \exists y \in V(S),\ m_{V_C}(y) = v\}$
23: **for all** pairs G_{B_i}, G_{B_j} for the set G_C **do**
24: Let $T_C = (E, V)$ be a new target composite shape such that $B_i, B_j \subseteq C$
25: Create a new *shape-level* map $M(C) \mapsto T_C$
26: Mark each newly transferred shape for this group G_C and those for the G_{B_i}, G_{B_j} pairs as conflicting with each other
27: **if** The set of faces for any shape is a subset of the set of faces of another shape with the same source analog **then**
28: Mark these two shapes as conflicting with each other
29: **return** M

slightly different figures in the target that overlap largely but not entirely—say, a single edge of a rectangle is in a slightly different location. As such, when two shapes with the same source overlap almost entirely, they are regarded as conflicting with each other. In particular, the algorithm tests the set of faces or cells, and looks for one being a subset of the other. Note that the "conflicting" relation is symmetric.

The result of this transfer process is a set of basic and composite that structure the target drawing (so to speak), and a mapping at the level of whole shapes from the source drawing's basic and composite shapes to the target's newly transferred shapes that may not be one-to-one. It is possible that not every newly transferred shape is proper, but this fact will not be obvious until the structural model is transferred, and so the removal of spurious shapes is left until then.

CHAPTER V

FROM STRUCTURE TO FUNCTION

Next up is model transfer: once the shape-level mapping has been inferred the task is to construct a model of the depicted device based on this mapping. Again the compositional analogy process moves up the levels of abstraction. The model being inferred is structural, causal, and teleological, and these three aspects must be inferred in that order. The shape-level mapping informs a structural model, an inference which is made possible by the two-level nature of the shape mapping (the basic and the composite shapes). These structural elements are the participants in causal processes, they move and interact with each other, and so from the structural model the causal (or behavioral) model can be inferred. Causal processes are the means by which the device's function is performed, and so from the causal model the teleological model, the device's function, can be inferred.

In this reasoning process, then, there are four levels of abstraction present: (1) the shape model, (2) the structural model, (3) the causal or behavioral model, and (4) the functional model—shape, structure, behavior, and function. The shape model has already been constructed by analogy, and now the reasoner has a mapping from the source to the target shape model. By analogy then it is to infer the structural, behavioral, and functional levels, moving from the more concrete level of depicted components to the more abstract level of device behavior and function.

I will take these steps one at a time, then: the transfer of structure, the transfer of behavior, and the transfer of function.

5.1 Transfer of Structural Elements

Each basic shape map in the shape mapping implies a component in the structural model, and each composite shape map implies a structural relation between components. Stated this way, the structural transfer algorithm is quite simple: iterate over the basic shape maps and transfer the depicted component, then iterate over the composite shape maps—which should be compositions of mapped basic shapes—and connected the already transferred components appropriately.

There are only two complications in this process. First, it may not be the case that all the components in a structural model are depicted. For instance, the crank case in the piston and crankshaft example is not depicted in any of the drawings that have been shown of this device. Thus, the structural transfer algorithm needs to extend the mapping to those components. Unfortunately, for any given component in the model there is no way to know *in advance* how many different components it may be connected to in a potential target analog. Thus, the algorithm at present simply transfers one of each non-depicted source component and establishes its connections appropriately.

The second problem is more complex. As it happens, the shape transfer algorithm sometimes transfers patterns of shapes that are redundant or otherwise spurious. This will result in structural transfers which are also redundant or spurious, and must be removed. They will generally have the property of being not properly connected to the rest of the structure, and so this step is called disconnected chain removal.

5.1.1 Disconnected Chain Removal

The problem of disconnected chains is best illustrated by example. Figure 31 shows a source and a target figure each with two overlapping rectangles and a circular arc segment. Imagine for the moment that the source figure has been divided into basic shapes in exactly this way, with two composite shapes: the pair of rectangles as one composite, and the rectangle *abcd* with the circular arc as the other. Surely, even divided into shapes in this way, there is only one map for each shape, but it turns out not to be the case.

The algorithms described int he last chapter will in fact transfer one big rectangle (the big rectangle on the left of each figure), one circular arc, and *three* of the little rectangles (*abcd* in the source). Specifically, the composite of the circlar arc and the rectangle *abcd* can map to the circular arc and one of three rectangles in the target:

- rectangle *uwxz*
- rectangle *vwxy*
- rectangle *uvyz*

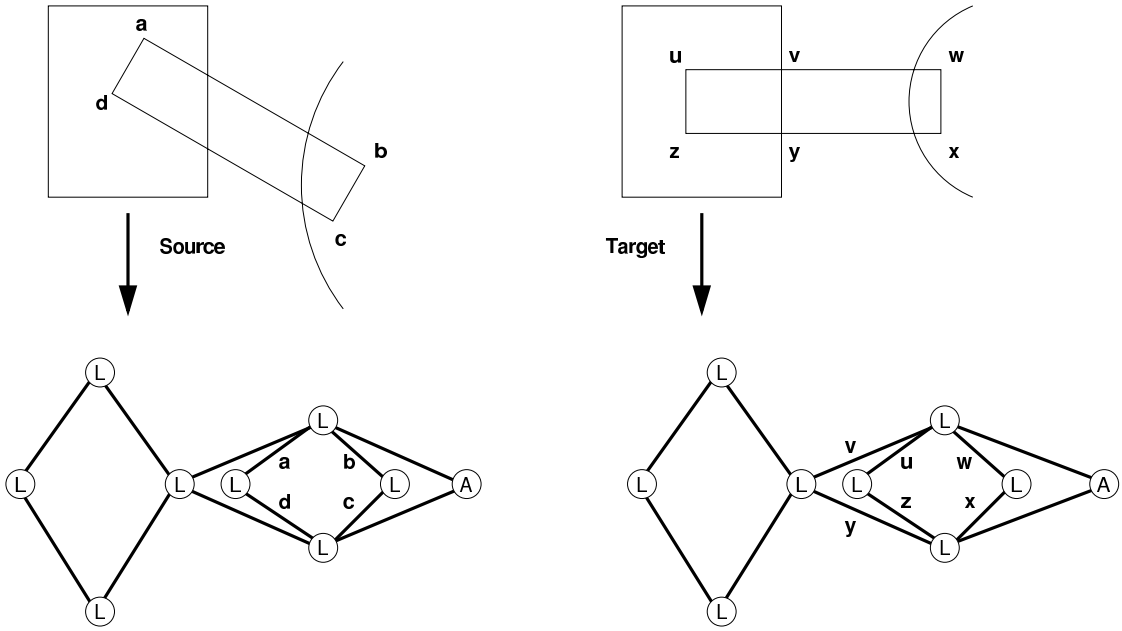


Figure 31: An example for disconnected chain removal: if the source on the left is divided into three shapes, then the central rectangle *abcd* will map onto three different rectangles in the target instead of the single rectangle one would expect

These are three *different* rectangles and are not symmetric with each other. However, the two rectangle composite maps only once onto the target, so only one of these—rectangle *uwxz*—is part of the other composite shape as it should be in the target. As such, if these basic shapes depicted three different components, then the resulting structural model would have one of the first component, one of the third, and three of the second, but only one of them connecting the first to the third as it should. The other two are *disconnected chains* and can be removed from the model, and this is precisely what the algorithm does.

Specifically, there are two conditions to consider. One is that certain shapes (specifically, the shape maps) are sometimes marked as *conflicting* with each other by the shape transfer algorithm. Whenever a pair of shapes conflict then one or the other is a candidate for removal. In the above example, for instance, all of the rectangles that *abcd* maps onto in the target overlap each other and will thus be marked as conflicting. All we need is a criterion for removing one or the other of them. The second condition, then, provides this criterion, and that is that each component to which a given component is connected should map to some component in the target that the is connected similarly to the given component's analog. That is, if the piston in the source is mapped to a piston in the

target, and the source piston is connected to a cylinder and also to a connecting rod, then the target piston should also be so connected.

To describe the algorithm precisely, I will introduce a notation for the shape and the structural model. Both can be regarded as graphs, and that is how I will treat them. Let M be a source model, and M' be the target model, with m a mapping between these models at several levels. Then $M[S]$ is the shape model, with $M'[S]$ the target shape model, and $M[C]$ the structural model. Each of these I will treat as a graph. The vertices of the shape model $V_{M[S]}$ are the basic shapes and the edges $E_{M[S]}$ the composite shapes. Likewise, the vertices $V_{M[C]}$ of the structural model are the components and the edges $E_{M[C]}$ the connections. Then the shape mapping is denoted $m_{M[S]}$ and the structural mapping $m_{M[C]}$. Note that these mappings are not mathematical functions necessarily, since they may map one source shape or component onto multiple targets, thus they are instead *relations*. Nevertheless, I will still write $m(x) \mapsto y$ when a source element x (shape, component, connection) maps onto a target element y, or else I will say $\langle s,t \rangle \in m$. With all this in mind, then, the algorithm for disconnected chain removal is shown in table 20.

5.1.2 Structural Transfer Algorithm

With this established, we can then move onto the structural transfer algorithm proper. The outline of the structural transfer algorithm is as follows:

1. *Transfer depicted components*: For each basic shape map, propose a new component in the target, and map the source component to this new target component

2. *Transfer depicted connections*: For each composite shape map, propose a new structural relation in the target, and map the source structural relation onto this new target structural relation

3. *Transfer non-depicted components*: For each *non-depicted* component in the source, propose *one* such component in the target, and again map the source to the target component

4. *Transfer non-depicted connections*: For each structural relation of a *depicted* and an *non-depicted* component, propose one such structural relation in the target for *every instance* of that *depicted* component in the target. For instance, if component A is non-depicted and

Table 20: The algorithm for disconnected chain removal

cleanup (m, M, M')

Input:	a mapping $m : M \rightarrow M'$ a source shape/structural model M a target shape/structural model M'
Output:	a revised shape mapping $m_{M[S]}$ a revised structural mapping $m_{M[C]}$ a revised target model M'

1: **for all** $\langle c, c' \rangle \in m_{M[C]}$ where c is a source component and c' is a newly transferred target component **do**
2: Let R be $\{\langle r, r' \rangle \in m_{M[C]} | r' = \{c', x\} \in E_{M'[C]}$ for some $x \in V_{M'[C]}\}$, the set of maps of connections for this target component c'
3: Let C be $\{e \in E_{M[C]} | e = \{c, x\}$ for some $x \in V_{M[C]}\}$, the set of connections in which the source analog for this component is involved
4: Let $s \in m_{M[S]}$ be the basic shape map for this component
5: **if** there is a connection $x \in C$ that does not have a corresponding connection map in R **and** the shape map s is marked as conflicting with any other shape map **then**
6: remove $\langle c, c' \rangle$ from $m_{M[C]}$
7: remove all of R from $m_{m[C]}$
8: remove c' from $V_{M'[C]}$
9: remove the basic shape depicting c from $V_{M'[S]}$, if any
10: remove all of the connections r' of c from $E_{M'[C]}$
11: for each removed connection r' remove the composite shape, if any, depicting that connection from $E_{M'[S]}$
12: **return** $m_{M[S]}$, $m_{M[C]}$, and M'

connected to component B, and A maps to A_1 and B maps to B_1 and B_2, connect A to *both* B_1 and B_2.

5. *Remove disconnected chains*: as described above, for each component in the target whose shape conflicts with another shape, check that it is connected to all of the same components (through the mapping) that it's corresponding mapped source component is, otherwise remove it and all of its structural relations.

6. *Complete the model*: transfer the quantities, quantity relations, and primitive functions associated with the transferred components

7. Return a component- and structural-relation-level mapping from the old, source structure to the newly instantiated target structure.

And so the output of this process will be a new structural model of the target and a mapping from the source to the target.

The only remaining detail to discuss is the last step: completing the model. This is straightforward: for each mapped component, transfer any quantities that it is related to by some quantity relation, transfer those quantity relations, and transfer any primitive functions associated with this component. This procedure is simplified substantially by the domain: the only quantities employed represent the motions of various moving components, there are no flowing quantities like water or heat. If there were, then the transfer of these quantities would be more complex and we could not simply iterate over the component mapping.

Since the model completion is straightfoward I will not state it in detail. In the structural transfer algorithm I represent this step as a function, complete(m, M, M') whose input is a source and target model, M and M', and a mapping $m : M \rightarrow M'$, and whose output is a structural mapping extended to include the transferred quantities and quantity relations (primitive functions are not mapped; there is no need, as will be clear when the behavioral transfer algorithm is described below), and the extended target model.

The full structural algorithm is shown in table 21.

Table 21: The algorithm for structural transfer

str-transfer (m, M, M')

Input:	a shape-level mapping $m = m_{M[S]}$ a source (shape and structural) model M a target (shape) model M'
Output:	a structural mapping $m_{M[C]}$ a target structural model $M'[C]$

for all $\langle s, s' \rangle \in m_{M[S]}$, $s \in V_{M[S]}$, $s' \in V_{M'[S]}$ **do**
  Let $c \in V_{M[C]}$ be the component depicted by the source basic shape $s \in V_{M[S]}$
  $V_{M'[C]} \leftarrow V_{M'[C]} \cup \{c'\}$, where c' is a copy of c, and let c' be depicted by s'
  Set $m_{M[C]}(c) \mapsto c'$
for all $\langle s, s' \rangle \in m_{M[S]}$, $s \in E_{M[S]}$, $s' \in E_{M'[S]}$ **do**
  Let $s = \{b_1, b_2\}$, where $b_1, b_2 \in V_{M[S]}$ are the basic shapes of which s is a composite
  Let $s' = \{b'_1, b'_2\}$, where $b'_1, b'_2 \in V_{M'[S]}$ are the basic shapes of which s' is a composite, and $m_{M[S]}(b_1) \mapsto b'_1$ and $m_{M[S]}(b_2) \mapsto b'_2$
  Let $r = \{c_1, c_2\} \in E_{M[C]}$ be the structural relation depicted by s, where $c_1, c_2 \in V_{M[C]}$ are the two connected components
  Let $m_{M[C]}(c_1) \mapsto c'_1$ and $m_{M[C]}(c_2) \mapsto c'_2$
  $E'_M[S] \leftarrow E'_M[S] \cup \{r'\}$, where $r' = \{c'_1, c'_2\}$
  Set $m_{M[C]}(r) \mapsto r'$
for all source components $c \in V_{M[C]}$ *not* depicted by a basic shape **do**
  Let c' be a copy of c
  Set $m_{M[C]}(c) \mapsto c'$
  for all $r \in E_{M[C]}$ where $r = \{c, x\}$ for some *depicted* component $x \in V_{M[C]}$ **do**
    Let $m_{M[C]}(x) \mapsto x'$
    Let $r' = \{c, x\}$
    $E_{M'[C]} \leftarrow E_{M'[C]} \cup \{r'\}$
    Set $m_{M[C]}(r) \mapsto r'$
$m, M' \leftarrow \text{cleanup}(m, M, M')$
$m, M' \leftarrow \text{complete}(m, M, M')$
return m and M'

5.2 Transfer of Behaviors and Function

The analogical transfer of behavior on the basis of a structural analogy is a subject that has been explored in the past. In SBF behaviors are represented *qualitatively* as we saw in chapter 3, and so the most relevant body of literature would be that of qualitative physical reasoning [19, 30, 57]. In particular, Falkenhainer's PHINEAS system [24] combined the structure-mapping engine [25] with qualitative process theory [30], and was capable of analogical inferences at the level of behavior on the basis of an analogies at the level of structure as well as behavior.

There are a number of differences between Archytas and PHINEAS that are important, not the least of which is the fact that Archytas operates at many levels of abstraction from drawings all the way up to function, whereas PHINEAS merely operates at the level of structure and behavior, but the most important difference is that the inferential task is in some ways more straightforward. PHINEAS made cross-domain analogies, and so it was necessary to have a qualitative physical reasoner for generating behavioral representations. However, in this application we have within-domain analogies and a complete structural mapping, so that it is not necessary to generate behaviors from scratch, but merely to transfer them over and make sure the causal relations are set up correctly. Not to mention, the state-transition model in SBF is far simpler, involving linear sequences of states rather than the splitting and branching that's possible with the behaviors generated in QPT—although of course the causal representations in SBF are more detailed, with many different kinds of causal relation represented that are not present in QPT. And finally, in SBF we explicitly represent function as well, which is completely absent (by design) from QPT and qualitative physical reasoning systems more generally, thus it will be necessary to transfer function as well as behavior.

5.2.1 Transfer of Behaviors

With a structural mapping in place, our task is now to transfer behaviors. Each behavior is a sequence of qualitative states of one of the components or quantities, and so the structural mapping directly informs the behavioral mapping to be constructed. However, there is a major complication: state transitions have certain causes that are represented in SBF (as we saw in chapter 3), and these

causes involve other behaviors and structural elements in the model, but when the *number* and *arrangement* of structural elements can change across an analogy, the network of causal relations must change as well. For instance, in a source model if a state transition in the behavior of component A causes a state transition in that of component B, and if A maps to A' in the target and B to both B'_1 and B'_2, then we cannot simply transfer the behaviors of A and B to the target. Does the state transition in the behavior of A' cause *both* the state transition in the behavior of B'_1 as well as that of B'_2? Yes, if A' is connected to both B'_1 and B'_2.

For a concrete example, take the piston and crankshaft example with the one piston source (figure 1(a)) and the double piston target (figure 1(c)). The downwards motion of the piston is caused by an external stimulus, but the upwards motion is due to the continuing rotation of the crankshaft as transmitted through the connecting rod. Thus, the transition of downward motion to upward motion occurs under the condition that the connecting rod so moves, which in turn occurs under condition that the crankshaft continues to turn, which it does so according to physical principles such as rotational inertia. But there are two pistons in the target, so for each piston, which state transition, which rod's behavior, causes its own? The answer is clear: the behavior of the rod connected to that piston.

More generally, we can say that only connected components can be causally related in their behaviors. Call this the *causal connection principle*. Following this principle, then, for each behavior we can calculate a *sphere of influence*, which is the set of other components to which that component is connected, the connections themselves, and all the properties, parameters, and primitive functions of all of the components. In addition, the sphere of influence includes all quantities that are related by quantity relations to the above set of components. When a behavior reference any of these elements (e.g. "under condition," or "using function," or what have you), it is assumed to be within that sphere of influence. The sphere of influence is calculated relative to the target, but in fact what we want are maps from the structural mapping.

5.2.1.1 Definitions

In order to give a formal description of the algorithm, once again it will be necessary to indroduce a notation. Recall the notation for the structural model above. Once again M and M' are the source

and target model, with $M[B]$ the behavioral model and $M[F]$ the functional model. Each behavior of some component $b(c) \in M[B]$ or of some quantity $b(q) \in M[B]$ consists of a sequence of states $s_i \in b$ and transitions $t_{i,j} \in b$ from a state s_i to a state s_j. For primitive functions, a given component C may have several functions, and so I will write $f_i(C) \in M[F]$ for each primitive function, and likewise $f'_i(C') \in M'[F]$ for the target.

The behavioral mapping is somewhat complicated by the need to map not only whole behaviors but individual states and transitions within those behaviors as well. I shall not go into too much detail in detailing this aggregate nature of the data structure, however, which seems clear, but simply say that if a behavior $b(c)$ maps to a target behavior $b'(c')$ then i shall write $m_{M[B]}(b) \mapsto b'$, or else $\langle b, b' \rangle \in m_{M[B]}$. For states, if s_i maps to s'_i I will simply say $m_{M[B]}(s) \mapsto s'$, and again for transitions $m_{M[B]}(t) \mapsto t'$. For the primitive functions, when $f_i(C)$ maps to $f'_i(C')$ I write $m_{M[F]}(f) \mapsto f'$ or again $\langle f, f' \rangle \in m_{M[F]}$.

5.2.1.2 Sphere of Influence

Given a target component c' whose source is c, we say that the sphere of influence of $\langle c, c' \rangle \in m_{M[C]}$ consists of:

- All the maps of structural relations r'_i involving c'
- Each map for the component c'_i to which a structural relation r'_i relates c'
- All of the maps of primitive functions of c'
- All of the maps of quantity relations qr'_i involving c' or some c'_i
- All of the maps of quantities q'_i from these quantity relations qr'_i

The sphere of influence of a quantity is that of all of the components to which it is related by a quantity relation. In the behavioral mapping algorithm, below, if $\langle c, c' \rangle \in m_{M[C]}$ is a component map, I shall write

$$C, R, F, Q, QR \leftarrow \text{influence}(\langle c, c' \rangle, m, M, M')$$

for the function that calculates the sphere of influence of a given component map. The algorithm is completely determined by the specification of the input and output, and so just this specification is

Table 22: The algorithm for the calculation of the sphere of influence of a component map. The calculation of the sphere of influence of a quantity is identical: it is the union of the spheres of influence of every component to which that quantity is related.

influence (cm, m, M, M')

Input:	a component map $cm = \langle c, c' \rangle \in m_{M[C]}$ a structural mapping $m = m_{M[C]}$ a source (structural and behavioral) model M a target (structural and partial behavioral) model M'
Output:	$R = \{\langle r_i, r'_i \rangle \in m_{M[C]} \mid r'_i = \{c'_i, c'\}$ for some $c'_i \in V_{M'[C]}\}$ $C = \{\langle c_i, c'_i \rangle \in m_{M[C]} \mid \langle r_i, r'_i \rangle \in R$ for $r'_i = \{c'_i, c'\} \in E_{M'[C]}\}$ $F = \{\langle f_i(c), f'_i(c') \rangle \in m_{M[F]} \mid f'_i$ is a primitive function of $c\}$ $QR = \{\langle qr_i, qr'_i \rangle \in m_{M[C]} \mid qr'_i = c'q'_i$ is a quantity relation in $M'[C]\}$ $Q = \{\langle q_i, q'_i \rangle \in m_{M[C]} \mid \langle qr_i, qr'_i \rangle \in QR$ for $qr'_i = c'q'_i$ in $M'[C]\}$

given in table 22.

5.2.1.3 Behavior Transfer Algorithm

With that, then we can state the algorithm. In outline it is as follows:

1. For each component map, if that component has a behavior, transfer the behavior of the source component to the target and map each state and transition

2. For each quantity map, if that quantity has a behavior, transfer the behavior of the source quantity to the target and map each state and transition

3. Go through the behavior maps and hook up the causal relations in the transitions of each behavior

4. Return the detailed behavioral mapping

table 23 shows the procedure used in step 3, and the complete algorithm is shown in table 24. The mapping of states to states and transitions to transitions is one-to-one, a direct transfer. The mapping of whole behaviors follows the pattern of the structural mapping, and again may be one-to-*many* rather than one-to-one. Individual causal links and slot values such as parameter values in states are not directly mapped, only whole states and transitions. This follows the structural transfer algorithm, which maps whole components but not their parameters.

Table 23: The algorithm for hooking up the causal links in the transitions of behaviors. This only shows the algorithm for component behaviors; the algorithm for quantity behaviors is identical. This is where the sphere of influence calculation is used.

hookup $(b, b'm, M, M')$

Input:	a source behavior $b(c)$ a target behavior $b'(c')$ a structural and behavioral mapping m a source structural and behavioral model M a target structural and behavioral model M'
Output:	a modified target behavior $b'(c')$

Let $cm = \langle c, c' \rangle \in m_{M[C]}$ be the corresponding component map
$C, R, F, Q, QR \leftarrow \text{influence}(cm, m, M, M')$
for all $\langle t_{i,j}, t'_{i,j} \rangle \in m_{M[B]}$ where $t'_{i,j} \in b'(c')$ **do**
    for all structural relations r in the *under-condition-structure* slot of the transition $t_{i,j}$ **do**
        Set the *under-condition-structure* slot of $t'_{i,j}$ to the target of all the connection maps in R that have r as a source
    for all components c_i in the *under-condition-component* slot of the transition $t_{i,j}$ **do**
        Set the *under-condition-component* slot of $t'_{i,j}$ to each target from the component maps in C with c_i as the source
    for all quantities q_i in the *under-condition-quantity* slot of $t_{i,j}$ **do**
        Set the *under-condition-quantity* slot of $t'_{i,j}$ to each target from the component maps in Q with q_i as the source
    for all transitions $\tau_{i,j}$ in the *under-condition-transition* slot of $t_{i,j}$ **do**
        Find the behavior for each transition, $\tau_{i,j} \in b(c_i)$ for some c_i
        Take all of the target behaviors for each component map $m_{M[C]}(c_i) \mapsto c'_i$ in C, $b'(c_i) \in M[B]$, and find the analogous transition, $\tau'_{i,j}$ from $m_{M[B]}$ for each
        Set the *under-condition-transition* slot of $t'_{i,j}$ to all of these transitions
    for all states s_i in the *under-condition-state* slot of $t_{i,j}$ **do**
        Find all analogous $s'_i \in b'(c'_j)$ by the same procedure
        Set the *under-condition-state* slot of $t'_{i,j}$ to all of these states
    for all Primitive functions $f(c)$ in the *using-function* slot of $t_{i,j}$ **do**
        Set the *using-function* slot of $t'_{i,j}$ to any target primitive function whose source in F is $f(c)$
return $b'(c')$

Table 24: The algorithm for behavioral model transfer. For consiseness this algorithm only shows the component behavior transfer method; the quantity transfer method is identical

beh-transfer (m, M, M')

Input:	a structural-level mapping $m = m_{M[C]}$ a source (structural and behavioral) model M a target (structural) model M'
Output:	a behavioral mapping $m_{M[B]}$ a target behavioral model $M'[B]$

1: **for all** $\langle c, c' \rangle \in m_{M[C]}$ **do**
2: **if** $\exists b(c) \in M[B]$ **then**
3: Let $b'(c') \in M'[B]$ be the behavior of c'
4: **for all** $s_i \in b(c)$ **do**
5: Copy s_i to s'_i
6: Add s'_i to $b'(c')$
7: Set $m_{M[B]}(s_i) \mapsto s'_i$
8: **for all** $t_{i,j} \in b(c)$ **do**
9: Copy $t_{i,j}$ to $t'_{i,j}$
10: Add $t'_{i,j}$ to $b'(c')$
11: Set $m_{M[B]}(t_{i,j}) \mapsto t'_{i,j}$
12: Set $m_{M[B]}(b(c)) \mapsto b'(c)$
13: **for all** $\langle b(c), b'(c') \rangle \in m_{M[B]}$ **do**
14: hookup(b, b', m, M, M')
15: **for all** $\langle q, q' \rangle \in m_{M[C]}$ **do**
16: **if** $\exists b(q) \in M[B]$ **then**
17: transfer $b(q)$ to $b'(q')$ by the same procedure
18: **return** m and M'

5.2.2 Transfer of Function

Although we may regard primitive functions as part of the functional model, they are closely associated with components and so needed in the behaviors and so are transferred as part of the structural transfer, as we saw above. Thus, the only remaining part of the functional model left to transfer is the functional specification of the whole device.

Since the representation of function is fairly simple in SBF, the transfer of function is straightforward. A functional specification, as seen in chapter 3, consists of a *given* and a *makes* state, and a link to a behavior by which the function is realized. Since these states are behavioral states, the transfer of function is simply a lookup of the states in the behavioral mapping.

The only problem with in this method is one-to-many mappings. For instance, the function of the piston and crankshaft is to turn the crankshaft, but the function of the piston and double crankshaft device is presumably to turn *both* crankshafts—but such a function is unexpressable in SBF, which requires the *given* condition to be a single state from a single behavior, and the *makes* state as well a single state from the same behavior. Since this function is unexpressable, we simply choose one of the behaviors and say that the function is to turn that particular crankshaft. A more satisfactory solution would require an extension of the expressability of the SBF representation, and so is beyond the scope of this work.

CHAPTER VI

EXPERIMENTS

As described in chapter 3, there were 26 drawings for Archytas, 5 source drawings and 21 targets, across three domains: that of the piston and crankshaft examples, that of the door latch examples, and that of the pulley examples. It was observed that the Common Lisp implementation in general was quite fast, taking on average less than 20 seconds for each example. Nearly all of that time in each case was spent in the shape analogy stage computing the individual mappings for composite shapes. In particular, these drawings represent differences in (1) the state of the device (e.g. the door latch pulled back versus pushed out), (2) differences of the dimensions of particular shapes (e.g. thinner versus thicker cylinder walls, longer versus shorter connecting rod, etc.), (3) differences of orientation in 2-D of the whole drawing (e.g. a 90° rotation or a mirror image of the original), (4) differences in the perspective in 3-D of the drawing, (5) differences of shapes depicting components (e.g. the cylinder as one U-shaped polygon versus two parallel rectangles), and even (6) differences in the number of components (e.g. 1(a) versus (c) or (d)).

In general, the success or failure of the shape analogy determined the success or failure of the whole process since the SBF models constructed tended to reflect very nearly the structure determined in this first stage. It was found that differences of device state, dimension, and orientation were straightforward and presented little difficulty. This is to be expected from the nature of the

Table 25: A brief summary of the differences between the source and target drawings that Archytas can and cannot handle: PC refers to differences tested with the piston and crankshaft example, DL refers to the door latch example, and PL refers to a third domain, that of pulleys.

Difference	**PC**	**DL**	**PL**
Device State	yes	yes	yes
Dimension	yes	—	—
Orientation	yes	—	—
Perspective	no	—	—
Component Shapes	no	no	no
Number of Components	yes	no	no

Table 26: Key to the identifiers used for each example in this chapter.

PC1 : First piston and crankshaft example
PC2 : Two piston and crankshaft example
PC3 : Piston and double crankshaft example
DL1 : First door latch example
DL2 : Double door latch example
PL1 : First pulley example
PL1a : Modified pulley example
PL2a : Four pulley example
PL3a : Pulley state change example
PL1b : Second pulley state change example

representation, as such changes do not alter the shapes at the augmented line intersection graph level. Changes in perspective or component shapes, however, were not handled, and changes in the *number* of components were sometimes handled well but sometimes presented difficulty. These were the most interesting cases that revealed the most about the methods. These results are shown in table 25. An analysis of these results in detail and their immediate implications will be explored in this chapter.

In the following discussion, 10 of the examples will be discussed in particular detail. These examples will be abbreviated in tables with a short identifier; table 26 is a key to the identifier used for each example in the tables that follow.

6.1 Evaluation of the Shape Transfer Methods

In the evaluation of shape transfer, the question is whether the "right" shapes are getting matched in the target drawing and transferred so as to enable transfer of the SBF model. For instance, in the piston and crankshaft example, we would like the rectangle representing the crankshaft in the source to match the rectangles representing crankshafts in the target drawings, but not those representing the cylinder or the piston. In order to measure this, it is possible here to borrow from the information retrieval literature and make use of precision and recall (see figure 32). A "query" now is a source shape pattern and the "responses" are the matching shapes in the target drawing. Measuring the degree of overlap of two sets—what should have matched a given shape and what actually did match a given shape—precision now asks, of those target shapes that actually matched,

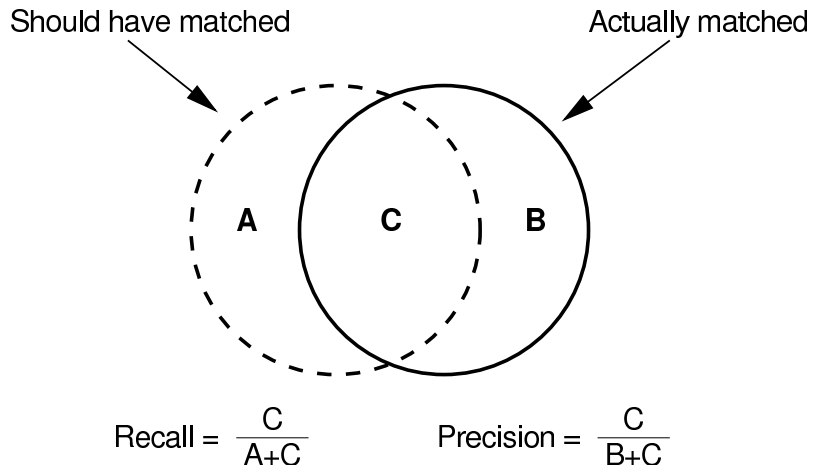


Figure 32: Precision and recall are standard performance measurements in retrieval systems, but can be generalized here for analogical comparison. It's possible to treat those forms in a target drawing that *should* have matched a given source shape pattern vs. those that *actually* did analogously to what should have vs. what actually was retrieved for some particular query in an information retrieval system.

Table 27: A count of the precision and recall in the shape mapping stage, for composite shapes only, broken up for each domains. Here, as per figure 32, column *A* counts the shapes that should have matched the drawing but did not, *B* counts those that did match but should not have, and C counts those that should have and did. Precision and recall are then calculated accordingly. In addition, B' counts the reduced value of *B* after disconnected chain removal, and thus "Precision-2" is the (hopefully) improved precision after removal.

	A	**B**	**B′**	**C**	**Precision**	**Precision-2**	**Recall**
Piston/Crank	0	2	0	14	88%	100%	100%
Door Latch	4	6	0	8	57%	100%	67%
Pulley	5	23	21	49	68%	70%	91%
Overall	9	31	21	71	70%	77%	89%

what proportion were correct; and, of those shapes in the target that should have matched, what proportion actually did?

Formulating our questions this way, looking at the matching of composite shapes alone (since the shape analogy stage iterates over the composite shapes in the source), we get the results shown in table 27. In addition, the disconnected chain removal stage of the structural transfer method also has the effect of removing the associated basic and composite shapes for the deleted components and connections. As such, from the perspective of precision, this should have the effect of (hopefully) increasing the calculated precision, and so table 27 also calculates a "Precision-2" from the modified

numbers after disconnected chain removal. This also evaluates to some extent the effectiveness of disconnected chain removal.

Overall precision was 70% before disconnected chain removal and 77% afterwards, a significant jump, while recall was 89% across all three domains. Looking at the individual domains, we see that the precision of 88% in the piston crankshaft examples became 100% after disconnected chain removal and the recall was 100%—perfect results for this domain. The other domains showed more interesting numbers, however. The door latch domain had the surprisingly low 57% precision which then jumped to 100% after disconnected chain removal—there were apparently many spurious composite shape matchings which got removed. The recall was, however, only 67%, so fully one third of the expected target shapes were not being found by Archytas. In the pulley example, the precision was 68%, going up slightly to just under 70% after disconnected chain removal—so clearly this removal is having little effect in this domain—while recall was 91%, better than the door latch domain.

A precision or a recall of less than 100% in a system like this causes great difficulty: the shape transfer algorithm supplies a shape-level mapping to the structural transfer algorithm, and if that mapping is incomplete or has spurious correspondences, then the corresponding structural elements will be missing or spuriously transferred. As such it is worth investigating these numbers in more detail to attempt to determine where and when the breakdown is occurring. Table 28 shows for each of the 10 detailed examples the number of shapes transferred, removed in disconnected chain removal, the total after this and the number expected (broken down for both composite and basic shapes this time—note again that the precision and recall numbers were for composite shapes alone).

The total and expected shapes for the three piston and crankshaft examples matched, as already established. The only redundant transfers were two extra shapes in the first example. In the door latch example, we see some extra shapes transferred in the first example (which showed a similar door latch in a different state) but then getting removed in disconnected chain removal, and so the totals are exactly as expected. The poor performance comes in the second door latch example, which showed the two latches connected to a single cam. In the pulley example, however, we see many redundant shapes being transferred, only getting things right in the last one.

It is worth investigating the door latch and pulley examples in more detail. Take first the double

Table 28: Composite and basic shapes transferred, removed during disconnected chain removal, the resulting total, and the number expected for each of the 10 examples. "Tr" stands for "Transferred," "Rm" stands for "Removed" (in disconnected chain removal), "Tot" stands for "Total," and "Exp" stands for "Expected."

	Composite shapes				**Basic Shapes**			
	Tr	**Rm**	**Tot**	**Exp**	**Tr**	**Rm**	**Tot**	**Exp**
PC1	5	2	3	3	6	2	4	4
PC2	6	0	6	6	7	0	7	7
PC3	5	0	5	5	6	0	6	6
DL1	6	2	4	4	6	2	4	4
DL2	8	4	4	8	10	4	6	7
PL1	23	0	23	13	24	0	24	13
PL1a	14	0	14	13	14	0	14	13
PL2a	19	0	19	15	16	0	16	14
PL3a	11	0	11	9	11	0	11	9
PL1b	6	1	5	5	7	1	6	6

Table 29: The results of running Archytas on figure 26(b): "tr" means shapes transferred, "del" means deleted, "tot" counts the total number in the final model (the difference between transferred and deleted), and the last column counts how many were expected in the "correct" answer. Shapes are named by the component or connection they depict.

Shape	**tr**	**del**	**tot**	**exp**
Cam/Door	0	0	0	2
Shaft/Bolt	4	2	2	2
Bolt/Door	0	0	0	2
Shaft/Cam	4	2	2	2
Door	0	0	0	2
Cam	2	0	2	1
Shaft	4	2	2	2
Bolt	4	2	2	2

Table 30: The results of running Archytas on figure 23(b), the columns are the same as the previous table, shapes are again named by the component or connection they depict.

Shape	tr	del	tot	exp
Rope-A/Wheel-A	1	0	1	1
Rope-B/Wheel-A	2	0	2	1
Rope-B/Wheel-B	2	0	2	2
Rope-C/Wheel-B	3	0	3	2
Wheel-A/F-Rope-A	2	0	2	1
F-Rope-A/Ceiling	4	0	4	1
Wheel-B/F-Rope-B	2	0	2	2
F-Rope-B/Weight	4	0	4	1
Rope-C/Ceiling	3	0	3	2
Wheel-A	3	0	3	1
Wheel-B	3	0	3	2
Ceiling	5	0	5	1
Fixed-Rope-A	2	0	2	1
Fixed-Rope-B	2	0	2	2
Rope-A	1	0	1	1
Rope-B	1	0	1	2
Rope-C	3	0	3	2
Weight	4	0	4	1

door latch example, the target in figure 26(b) (page 55), with 20 (page 46) as the source. Table 29 shows the number of basic and composite shapes transferred, removed, and expected for each shape. We see that the cam/door composite and the bolt/door composite had no matches at all. Among the basic shapes we can see the cam and the bolt did match (in the other composite shapes) and so the problem was obviously the door. In this case, the shape of the door *changed* from source to target: instead of a C-shaped outline with two rectangles representing the plate through which the bolt moves, there were two such rectangle pairs on either side and the outline was no longer C-shaped. No match was possible and so no match was found. Although this merely demonstrates results already summarized above, it is an interesting and instructive illustration of the limitations of the method.

The pulley examples were more involved. Here the problem appeared to be *too many* matches rather than too few. We see in the first pulley example, table 30, that the pulley wheels caused particular problem, with six total transferred, and rope C (going from the lower pulley to the ceiling) and the weight also were ambiguous shapes. Thus, many more were transferred, and, curiously,

Table 31: The results of running Archytas on figure 28(b), the columns are the same, shapes are again named by the component or connection they depict. Rows with expectation failures have been italicized.

Shape	tr	del	tot	exp
Rope-A/Wheel-A	1	0	1	1
Rope-B/Wheel-A	2	0	2	1
Rope-B/Wheel-B	2	0	2	2
Rope-C/Wheel-B	3	1	2	2
Wheel-A/F-Rope-A	2	0	1	1
F-Rope-A/Ceiling	4	0	1	1
Wheel-B/F-Rope-B	2	0	1	2
F-Rope-B/Weight	4	0	1	1
Rope-C/Ceiling	3	0	2	2
Wheel-A	3	0	2	1
Wheel-B	3	0	3	2
Ceiling	5	0	1	1
Fixed-Rope-A	2	0	1	1
Fixed-Rope-B	1	0	1	2
Rope-A	1	0	1	1
Rope-B	1	0	1	2
Rope-C	3	1	2	2
Weight	4	0	1	1

none whatever were removed in disconnected chain removal. Looking at the drawing, we see first of all that the pulley wheels are drawn as a small rectangle overlapping a circle. This rectangle is potentially ambiguous with the weight, which is also a rectangle, hence the four weights in the drawing instead of one.

This lead to the second set of pulley examples, with Pulley-A as the source (figure 28(a), page 57, is the source drawing). The source drawings are hard to distinguish, but the two changes are the weight is drawn differently and the rectangle overlapping the circles for the pulley wheels is no longer a rectangle. This eliminates many of the redundancies, as shown in table 31. Once again we see some redundant transfers but virtually nothing removed in disconnected chain removal. In particular the wheels were transferred too many times, fixed rope B (going from the second wheel to the weight) was only transferred once when it should have been transferred twice, and rope B (going between the first and second wheels) was again transferred once instead of twice.

These same trends: for ropes to be transferred too *few* times and wheels too *many*, were confirmed in the second two Pulley A targets: figure 28(c) (page 57) and the second one was the original drawing (figure 28(a)) itself. There are two problems illustrated through these examples:

1. The wheels are two separate components of the same component *type* (a pulley wheel), but the SBF language does not distinguish component types from individual components, and so each wheel in the source matches each wheel in the target resulting in too many transfers.

2. A rope is just a straight line, which should match any straight line, but in Archytas a line is defined by its intersection points, and the intersection of a line with another line (such as the rope with the ceiling rectangle) is different from the intersection of a line with an arc (where a rope meets a wheel), and so whenever pulley wheels are added to the target drawing the ropes between the wheels will sometimes fail to match—in particular, when a rope that went (in the source) from a wheel to the ceiling (say) goes now (in the target) from a wheel to another wheel.

Bringing these two observations together we might say that a kind of visual symbol mismatch is occurring, *A* and *B* are different symbols and so they must be different even though the same pattern can be seen in both. Were the pattern examined, it would be shown to be the same, but the pattern is not examined.

Bringing together the observations from this section, then:

- Shape matching was robust against changes in the length and width and orientation of shape patterns, as expected

- Before the structural transfer stage, 70% of the composite shapes that matched were correct, the other 30% were spurious.

- Disconnected chain removal significantly improved the shape matching, removing many spurious matches and bringing precision up to 77%

- 89% of the potential composite shapes in target drawings that should have been found were actually found; the remaining 11% were simply not found

- When redundant shapes are transferred to overlapping patterns, this can be detected and removed
- When component shapes change significantly in visual structure between source and target, they fail to match the target
- Redundant shape and therefore structural transfers are not always removed if the resulting model elements are still connected to each other in some fashion
- A variation of the symbol mismatch problem occurs with shapes: wheel-A and wheel-B are clearly both wheels, and so when one pattern matches them both that indicates one wheel and not two; likewise, rope-A and rope-B are both ropes, and when they match the same line segment, that indicates one rope connecting two wheels, not two disconnected wheels

The implications will be treated in chapter 8.

6.2 Evaluation of the Structural Transfer Methods

In the transfer of structure, the algorithm iterates over the shape mapping provided by the previous stage. As such, if the shape mapping is correct, then the structural transfer will ipso facto be correct as well. Hence, there were really just two things evaluate:

- The transfer of non-depicted components
- The removal of spuriously transferred components and connections

The first of these is not as complex an issue as it might seem, and its limitations in Archytas were plainly established by the piston and crankshaft examples (the only ones that had non-depicted components to transfer). In particular, in the third example (PC3, the piston and double crankshaft example, figure 1(d), page 2), there were two crankshafts, but Archytas as always transferred the crank case only once. The question is: should there have been one or two crank cases in this example? It's not clear on the face of it what the answer is, and at any rate it must involve more complex model-based reasoning than Archytas engages in. With a more complex method for dealing with non-depicted components, the evaluation would necessarily be more complex, but as such there is a single limitation and this example illustrates it.

The second of these is more interesting, and the results can already be seen from the previous section. In particular, we see in the first piston and crankshaft example (figure 1(b)) the disconnected chain removal working perfectly, as it does in the first door latch example. In the second door latch example, insofar as some of the components were transferred correctly, redundant transfers were removed successfully, and so there were no remaining redundancies except the pair of cams. This is interesting: the component changed its shape as a result of performing two roles instead of simply one (that of moving two shafts), and so the shape overlapped itself twice and hence matched twice. That the shapes overlapped means nothing in Archytas and so two components were transferred. The missing inference, that these are really one component, is really a very subtle inference and it's not simply clear when it should and should not be made.

In the pulley examples we see the disconnected chain removal doing almost nothing at all. In particular, the method of disconnected chain removal is to iterate over the structural mapping and perform a single test on each component—is it connected to all the other components that it should be (i.e. that its source analog is)?—and to remove those components that fail the test. It should be clear that this method is order sensitive since it goes through the list only once, and furthermore that there could be interactions (e.g. a disconnected loop of components in which each individual one appears to be well connected when the set of them is not). The problem of disconnected appears, then, to be more complex than previously assumed.

6.3 Evaluation of the Transfer of Behavior and Function

6.3.1 Interdependency of the Stages

Table 32 shows whether the shape, structure, behavior, and function were transferred correctly for each model. Before delving into the reasons for correct and incorrect transfers of behavior and function, it is worthwhile to investigate some interesting patterns in this table:

- Of the five models in which a correct shape model was transferred, four of them (80%) also had a correct structural model.
- Given a correct structure, a correct behavior was always transferred
- Given a correct behavior, a correct function was always transferred.

Table 32: For each example, this table shows whether each of the four models—shape, structural, behavioral, and functional—were transferred correctly.

	Shape?	Structure?	Behavior?	Function?
PC1	yes	yes	yes	yes
PC2	yes	yes	yes	yes
PC3	yes	no?	no	yes
DL1	yes	yes	yes	yes
DL2	no	no	no	yes
PL1	no	no	no	yes
PL1a	no	no	no	yes
PL2a	no	no	no	no
PL3a	no	no	no	yes
PL1b	yes	yes	yes	yes

- Given an *incorrect* shape model, not a single case correctly inferred the structure.
- Given an incorrect structure, not a single case correctly inferred the behavior.
- Given an *incorrect* behavior (there were six of these cases), fully five of the six *correctly* inferred the function.

The lone exception to the correct structure from correct shape—PC3, the piston and double crankshaft example—is an ambiguous case and it could be argued that the structural model was as close to "correct" as can be expected within the limits of the representation and the mode of inference. However, the most interesting point is the last one: function was correctly inferred in 9 out of the 10 examples. As it turns out, since function is an abstraction of behavior, and in particular an abstraction of a particular behavior, function could be correctly transferred so long as that particular behavior had been transferred, regardless of whether the rest of the behavioral model was correct or incorrect.

With that interesting and notable exception, these results establish a very clear interdependency in the stages of the compositional analogy process. In particular, the clearest predictor of success was the success of the first stage: if the inferred shape model is incorrect, nothing following it is correct. This is to be expected in a method that relies entirely on analogical transfer as its primary mode of inference; we could only expect failure recovery if there were some other (model-based) mode of inference that could determine when incomplete or incorrect models had been inferred.

Note, however, that the above results are those *after* the so-called disconnected chain removal has run. If we note that two of the supposedly correct shape models were in fact *incorrect* until this stage had run (which is part of structural transfer, recall), then some of the above results change slightly. In particular, there are now *seven* incorrect shape models, and of these, two of the seven cases (29%) saw a correct structural model inferred from an incorrect shape model.

6.3.2 Evaluation of Behavioral and Functional Models

So far, we have only asked whether the entire behavioral model was correct or incorrect. But in general in each case there were particular behaviors correctly inferred and others incorrectly inferred or not inferred at all. The question is, then, which behaviors were correctly inferred, which were not correctly inferred, and why?

Across the ten examples, there were 84 target behaviors transferred from 50 source analogs (counting per source/target pair, so for instance the piston and crankshaft source with 4 behaviors was used for three targets, transferring 4 behaviors to the first target, 7 to the second, and 7 to the third; this makes for 12 source analog behaviors—rather than 4—and 18 target analogs). Of these 84 behaviors, 56 were correct (67%). Compare this statistic with the 4 out of 10 examples with completely correct behavioral models from above, indicating that in many examples partially correct behaviors sometimes did occur. By domain:

Piston and crankshaft:	15 out of 18 correct (83%), from 12 source analogs
Door latch:	8 out of 9 correct (89%), from 6 source analogs
Pulley:	33 out of 57 correct (58%), from 31 source analogs

Once again, the pulley example shows the poorest performance. Of themselves these numbers mean very little; mostly they show that the biggest problems lay with the pulley examples. However, they also show that despite the long string of incorrect behavioral models in the pulley examples (4 of 5), more than half of the transferred behaviors were in fact correct. As it turns out, every example transferred at least a few behaviors correctly. The worst one, the first pulley example (PL1) saw 7 of the 23 (total, 11 expected) behaviors transferred correctly.

Next we turn to an examination of the transferred behaviors themselves. The piston and crankshaft domain had three examples, of which the first involved a change of state and therefore no difference

between source and target at the structural level, and therefore none at the behavioral or functional levels either. The second and third examples involved changes in structure that implied substantial differences at the behavioral level. In the first of these cases, the dual piston and crankshaft example (PC2), the change in structure from source to target involved duplicating part of the structure (the piston, cylinder, and connecting rod) and connecting it up to the remaining part (the crankshaft and crankcase) which was not duplicated. Here the correct inferences involved transferring the relevant behaviors twice and hooking up the causal links (the "under-condition" and "using-function" slots) correctly—in particular it involved disambiguating which piston was causing which connecting rod to move, and that *both* connecting rods were causing the one crankshaft to move (and in turn in the second half of the behavior that the one crankshaft was causing each connecting rod and piston to move back up the cylinder). This it did correctly:

Piston 1 Linear Motion:	correct
Piston 2 Linear Motion:	correct
Connecting Rod 1 Linear Motion:	correct
Connecting Rod 1 Angular Motion:	correct
Connecting Rod 2 Linear Motion:	correct
Connecting Rod 2 Angular Motion:	correct
Crankshaft Angular Motion:	correct

The last piston and crankshaft example, the piston and dual crankshaft (PC3) was interesting. Again it involved a duplication of part of the structure (the connecting rod and crankshaft) but not the rest. Here, however, the correct behavioral inferences were not made. In particular, while the behaviors of the piston and first connecting rod and crankshaft were all transferred correctly (unmodified), the second rod and crankshaft behaviors were not transferred correctly:

Piston Linear Motion:	correct
Connecting Rod 1 Linear Motion:	correct
Connecting Rod 1 Angular Motion:	correct
Crankshaft 1 Angular Motion:	correct
Connecting Rod 2 Linear Motion:	*incorrect*
Connecting Rod 2 Angular Motion:	*incorrect*
Crankshaft 2 Angular Motion:	*incorrect*

Here something interesting has occurred with the transfer: the direction of causality has changed. In particular, now a connecting rod is connecting one crankshaft to another rather than a piston to a crankshaft, which means in the first half of the motion of connecting rod 2, the first crankshaft is doing the turning, and in the second the rotational momentum of both crankshafts is continuing the motion. The method of making causal connections across the behavioral transfer is the sphere-of-influence calculation, and here this calculation failed it: the downward motion of the connecting rod is caused by the piston, but in the second connecting rod it should be caused by the first crankshaft, something not present in the source as a cause. This reversal in causation is beyond the capacity of a mere transfer algorithm, and requires some capacity for physical inference in order to observe that the new pattern of structural relations has different implications than the source does with regard to the sequence of events and their causal connections.

In the first door latch example (DL1) we again observe that the behaviors were transferred correctly for this case in which the only substantial change is a change of state:

Cam Behavior:	correct
Shaft Behavior:	correct
Bolt Behavior:	correct

More interesting was the second door latch example, the double latch, in which there was one cam, but two shafts and two bolts:

Cam Behavior: *incorrect*

Shaft 1 Behavior: correct

Shaft 2 Behavior: correct

Bolt 1 Behavior: *incorrect*

Bolt 2 Behavior: *incorrect*

The bolt behaviors were in fact correct except for one small but very critical detail: the door was missing from the structural model, and with it the structural connection between the bolt and the door as a structural cause of the movement of the bolt. Since the bolt moves through a prismatic joint in the door, this is an important omission. Except for this, the behaviors were correct. The behavioral cause, that of the movement of the shaft, was correct for each bolt's behavior. The shafts of course were transferred and connected appropriately and their behaviors were correct. The cam behavior was incorrect as Archytas transferred *two* cams instead of one (as discussed above) and thus two cam behaviors. Furthermore, since again the door is missing and the cam turns in a revolute joint in the door, this part of the cause of the cam's motion was also missing. In short, once again the principle problems were those of (a) number (two cams instead of one) and (b) the omission of a key structural element (the door).

In the pulley examples I will refrain from listing each expected behavior as above, as there were forty-four expected behaviors across the five examples (PL1, PL1a, PL2a, PL3a, and PL1b). It's easier to list the correct behaviors, as these were few:

- Rope A behavior in all examples (the rope at the far right of the drawing, whose pull initiates the motion of the pulley)
- Weight motion in all examples except PL2a (and in PL1, while one of the weight motions was correct, there were four transferred and the other three were of course incorrect)
- All of the expected behaviors in PL3a (which matched the two pulley drawing with itself), except that for each of the pulley wheels two behaviors were transferred where one was expected, and hence one of these two behaviors was of course spurious and so incorrect
- All of the behaviors in PL1b

All other behaviors in the pulley example were incorrect. In most cases, there were either too many or too few behaviors transferred (the result of either too many or two few components in the structural model). In other cases the lack of correct behaviors meant that causal links in other behaviors were incorrect, so this transference of incorrect behaviors has a cascading effect down the chain of causation, to some extent. The exception was the weight: since in all examples some behavior for fixed rope B (which connects the weight to the pulley wheel directly above it) was transferred, therefore the causal antecedent for the weight was transferred, and since at least one correct weight was transferred in all but one example (PL2a), the motion of the weight was correctly inferred in 4 of 5 pulley examples.

Summarizing then, the following observations can be made regarding behaviors:

- If there is no change in structure (and the inferred structural model is correct), behavior can be inferred correctly
- If there is an addition in causes as a result of a change or addition to structure—such as two connecting rods causing one crankshaft to turn, or one crankshaft causing two connecting rods to move upwards—then the correct behavior can be inferred
- If there is a change or reversal in causation—such as a crankshaft causing a connecting rod to move downwards rather than a piston being the cause—the correct behavior is not inferred
- If the structural model is incorrect, with missing components, the correct behavior is not inferred
- Correct behaviors cannot be inferred for spuriously transferred components

And finally there is the question of functions. There is little here to say, since function is in all cases in these domains an abstraction of behavior with an input and an output state. These states are always (in these examples) states of some one particular behavior, so as long as that particular behavior was correct, there was no problem with function. This was true in all cases except one, that of PL2a, where the shape of the weight did not match. In this example, the angle between the pulley bracket (the rectangular shape over the pulley wheel) and fixed rope B was perpendicular in the source, and so Archytas expects this angle to be perpendicular in the target but it is not for this

image and so the weight is not transferred. With no weight transferred, no behavior was transferred and so no function.

The only other issue for function is that in two of the examples (PC3 and DL2) the component involved in the output behavior was duplicated between source and target—the second crankshaft, the second bolt in the door latch—and it's not immediately clear what the resulting function should be. The limitation is that a single state must be specified in the input and output states of the function, and a single behavior for both of these states, and a single behavior cannot be one of multiple components at once; rather there are two behaviors that are conceivably both involved in the function. As such, the function was considered "correct" if it referred to a correct behavior for either of the two possible behaviors, and this was indeed the case in both PC3 and DL2.

CHAPTER VII

RELATED WORK

In examining other work related to my theory of compositional analogy, quite a diverse range of subjects needs to be explored because of the vertical nature of the methods. That is to say, many theories in artificial intelligence tend to spread themselves out horizontally rather than vertically, an example being a problem solver such as a forward-chaining reasoner or a STRIPS planning system intended to apply in a wide range of domains once the domain knowledge is encoded in some ontology or rule set. However, such theories very often support only a narrow range of kinds of inferences, such as a system that can support deductive chaining but not abductive or inductive inference, or a system that can support mathematical reasoning across many domains but not physical reasoning or vice versa.

By contrast, a few theories in AI and cognitive science extend vertically rather than horizontally, covering a single task or a narrow range of tasks from the original stimulus to the final output, through several layers of reasoning, and thus necessarily support many kinds of inferences to cover the range of different kinds of knowledge employed in the task. Compositional analogy—and its implementation in the form of Archytas—is an example of this sort of theory. It begins with a drawing as input and outputs a detailed teleological model of what the drawing depicts, proceeding through several layers of reasoning and levels of abstraction along the way.

This vertical integration means that a wide range of subjects can be brought to bear on the problems with which this work is concerned. Here I divide up the related work by that part of the method to which it relates. Once again I have divided the basic problem of this work into two halves: inferring a structure from a drawing, and inferring a functional representation from a structure.

7.1 From Drawing to Structure

7.1.1 Visual and Diagrammatic Reasoning

Diagrammatic reasoning is a burgeoning field at the intersection of several other fields, most notably the traditional cognitive science influences of psychology and computer science, but also architecture and other design-related disciplines, as well as education. The literature in this field is vast, and beyond that it is a difficult field to circumscribe, because it is rather new and crosses several disciplinary boundaries, with papers appearing in scattered conferences and journals all over the various cognitive sciences, and often in the design literature [40, 42, 41], though there have been some recent attempts to unify the field [51, 8, 5]. Here I will only spotlight a few notable examples of work in the area.

Much of the work in this field has been psychological, rather than from an AI perspective. Some of this is interesting in inspiring cognitive models, but this dissertation does not represent an attempt to do cognitive modeling. Nevertheless, there is plenty of relevant work on computational models and approaches. Broadly speaking, the idea behind the focus on diagrammatic reasoning is to study the use of diagrams as a kind of external "analogical" representation, in contrast to so-called "Fregean" or logical representations [70]. Within AI, the attempt is to find ways of solving interesting problems by representing them externally in a diagram using location, shape, relative position, and so on, rather than in some more conventional symbolic language. In practice, of course, a computer program can only work with symbolic representations, and so in fact it is representing a diagram symbolically, and abstracting the relevant information from the diagram and translating into a more conventional symbolic representation. As such, in practice, so-called diagrammatic reasoning is really employs another *mode* of representation in problem solving, so that it might just as well be called *multi-modal* representation and problem solving (and it often is).

One of the first contributions to this literature is the oft-cited paper of Larkin and Simon [60], who describe a system that could reason about pulley systems, among other things. In this system, to answer a question about a pulley system, the reasoner uses a bit of non-visual physics knowledge along with knowledge of location, and all information associated with locations, rather than an object-based language like you might see in a qualitative physical reasoner.

Another early system was Funt's WHISPER [35], which had a simple retina model with higher

resolution towards the center, and solved blocks world problems in a visual manner.

The work of Alvarado and Davis [1, 2, 3] on sketch-based recognition in the SketchREAD system was discussed in chapter 2. In addition to this, Randall Davis has done much work on the subject of sketch-based recognition in user interfaces [16, 69]. The focus for much of this work has been on the implications for user interface design and human-computer interaction. In terms of AI, much of the work has been on methods for disambiguation of low-level shape patterns, e.g. when a pen stroke can be regarded straight line and when it can be regarded as a curve, and the models involved have tended to be rather simple.

A related project is the SketchIT system of Stahovich, Davis, and Shrobe [71], a sketch recognition system that reads in a sketch of a simple design and produces several variations on that same design as output. The sketch is augmented with a simple state-transition diagram to describe the device's behavior. The system produces a so-called *behavior-ensuring parametric model* (BEP-Model) of the device, and from this determines geometric constraints on the motion of the parts, generating all *qc-spaces* (qualitative configuration spaces) consistent with the behavioral constraints. From this, motion types for each component are selected, and geometric interpretations provided by a library of interaction types, and from this a new BEP-model is generated of the new design. The system only handles rigid bodies and springs, and the models only handle geometric constraints on motion, and are not physical models of motion in any sense. However, more recent work [59] uses visual symbol recognition (which must be trained) and behavioral reasoning to identify intended meaning, not just behavioral constraints, and can deal with motors, heaters, pulleys, wheels, and so on. The new system produces a textual gloss describing the behavior or functioning of the sketched device at a high level ("lever rotates", for instance) based on a simple qualitative physical simulation, which is achieved with a forward-chaining rule system. This physical model is very simple (e.g. it does not distinguish direction of rotation, or direction of force) and is only used to disambiguate the roles of parts in functioning of the device.

More generally, sketch-based recognition work has become rather popular with the rise of pen-based interfaces [17]. Some interesting work has come out of this effort, most of it focused on low-level bottom-up processing such as recognition of freehand strokes and basic visual symbols. Some of the work is model-based, such as with SketchIT and SketchREAD, but the underlying

models used to interpret the sketches are generally quite simple, and this is intentional, since the bulk of the work has surrounded issues of ambiguity with regard to users' intention, such whether a certain stroke was intended to represent a square or a circle. This has lead to some interesting work on multi-modal interfaces, for instance involving both sketch-based and vocal input.

7.1.2 Matching Relational Structure

The first part of compositional analogy, inferring a structure from a drawing, is akin in certain ways to object recognition. Suetens, Fua, and Hanson [76] survey a number of methods for object recognition, but the most directly relevant of the methods discussed are the so-called "structural pattern recognition" methods, where "structure" here means graph structure (in the graph-theoretic sense), or relational structure in a knowledge representation—these are of course two different ways of regarding the same problem.

From a mathematical standpoint, such methods reduce the problem of object recognition to one of subgraph isomorphism. This is a well-explored problem in computer science. One of the earliest papers on the subject is Ullman's algorithm for subgraph isomorphism [80]. Eppstein [22] cites some more recent results from the graph algorithm literature. In his paper, discussed in chapter 4, Eppstein showed that for a fixed pattern, when the graphs in question are *planar* graphs, then it is possible to compute the match in *linear* time, though (again as discussed in chapter 4) his result is not directly applicable in this work.

In chapter 4, I reduced this structural alignment to a constraint satisfaction problem, building on previous work [86]. In doing so, I have only just scratched the surface of constraint satisfaction. Constraints in general are relationships between the variables of the CSP—or more precisely their domains—and so it should come as no surprise that the structure of these relationships should go a long ways towards determining the difficulty of the problem [33, 34, 45]. In particular, certain kinds of patterns of relationships can be taken advantage of by such techniques as back-jumping or dependency-directed backtracking, or forward checking. In a normal backtracking algorithm, such as that employed by Archytas, which is simply a plain depth-first search, when a conflict between variable assignments is found, the method backtracks one variable at a time until the conflict is resolved. However, in dependency-directed backtracking, when a conflict between assignments is

found, the method backtracks to the conflicting variable, and this can sometimes reduce the size of the search quite considerably. In forward checking, when a value is assigned to some variable, potentially conflicting values are eliminated from the domains of variables yet to be assigned, so that the method backtracks when the domain of a variable is empty—this too can reduce the size of a search in some circumstances. Kondrak and Van Beek [55] give a theoretical evaluation of several such methods, and in particular they find an ordering of the methods based on how much of the search space (of that explored by basic backtracking) each method will explore for any given constraint satisfaction problem.

7.2 From Structure to Function

7.2.1 Qualitative Physics

In the application domain of kinematics devices, or even engineering design more generally, I know of no other system that directly infers teleology as well as causal process from structure alone, analogically or otherwise. Having said that, there is a substantial body of well established work on the subject of inferring causal processes from structure, and that is qualitative physical reasoning.

In qualitative physics, the problem is to take a qualitative description of a physical device or situation and infer aspects of its behavior. The goal is to generate a *qualitative envisionment*, in which a more or less complete discrete state-transition model of the behavior, from which other inferences can be made (it is discrete, of course, due to the qualitative nature of the representation). What is involved in this envisionment can vary. The Qualitative Physics based on Confluences of De Kleer and Brown [19] as well as Forbus's Qualitative Process Theory (QPT) [30] both make it a goal to directly represent causation to at least some degree. If wheel *A* causes wheel *B* to turn, this is different from wheel *B* causing wheel *A* to turn, and in these theories the distinction is important. By contrast, in Kuipers' Qualitative Simulation (QSIM) [56, 57], causality is represented only insofar as a differential equations represent causality, and the primary goal is to show that the *qualitative differential equation* (QDE) is a precise mathematical abstraction of a differential equation, and thus QSIM derives qualitative behavioral trajectories from qualitative mathematical constraints.

Thus from these three bodies of work we see the three basic proposals for qualitative physical reasoning: (1) the qualitative equations of De Kleer and Brown, based on a sign algebra (e.g.

two positive quantities added together make another positive quantity, but a positive and a negative quantity added together are indeterminate); (2) the qualitative proportionalities and qualitative influences of Forbus's QPT, which make up the qualitative processes and views, from which behavior is inferred; and (3) the qualitative constraints of Kuipers' QSIM, which make up the qualitative differential equations from which behavior is inferred. All of these differ in organization and in underlying philosophy from SBF, which explicitly attempts a functional decomposition of a design for analogical inference rather than qualitative physics' commonsense representation of behavioral constraints from which behavior might be inferred from structure in a deductive manner.

This has several implications. First, qualitative physical models will typically place causal relationships between variables in the structural model, whereas in SBF such relationships are considered part of the behavior—i.e. the causal processes themselves—and are not considered structural in nature. Qualitative physics is organized in this way in order to allow causal inferences to be drawn in a bottom-up fashion, just as SBF is organized in its way so as to support functional design in which the reasoning is exactly backwards, starting from function and proceeding *downward* to structure. Second, SBF explicity represents function as an integral part of behavior and even structure to some extent, whereas qualitative physics, for the sake of inferential completeness, must not allow function to be represented at all (the so-called "no function in structure principle"). Third, SBF can support functional reasoning in design whereas qualitative physics cannot, but conversely qualitative physics can support predictive reasoning about the behavior of a given device whereas SBF cannot, and in fact has never been used in such a manner. Finally, it is important to note that qualitative physical reasoning is deductive in nature rather than analogical, whereas SBF has been developed as part of an explicitly analogical theory of causal and teleological reasoning. That is, it may be possible to use analogy to perform what would otherwise be a deductive inference, and likewise it may be possible to produce a logic that provides a deductive closure for what would otherwise be an analogical inference, but in general the goals of deductive versus analogical inference are opposed to each other and call for different kinds of knowledge to be organized and employed in different ways.

Much of the work on qualitative physics has been based on flow models, and the theories have tended to work well whenever the device in question can be described in terms of flow (e.g. water

through pipes, or the spread of heat). But in mechanical domains, qualitative inferences often depend on *quantitative* knowledge. For instance, two rollers, one with a notch and one with a nob or tooth fitting the notch, may roll together when they are turned at a certain precise angle with respect to each other, but not otherwise, and so the inference of behavior requires knowledge of a specific quantity, in this case an angle, which is unavailable in a qualitative model. In the *metric diagram/place vocabulary* (MD/PV) theory of Forbus, Neilsen, and Faltings [31, 32], which was explored in the CLOCK project, this sort of knowledge is supplied by the *metric diagram*, from which is inferred a *place vocabulary* that relates the diagrammatic knowledge to the physical model. In the CLOCK project, the goal was to build a qualitative physical reasoner capable of reasoning about kinematics devices, and the particular example was a ratchet mechanism in a clock (hence the name). The system took a diagram of the ratchet mechanism along with a representation of the constraints on the motion of particular shapes in the diagram and attempted to generate a qualitative envisionment of the device's behavior.

Within this project, Faltings's work [26] focused on the generating of place vocabularies from metric diagram information and configuration spaces, which represent constraints on possible motions of parts. Nielsen's work [63] focused on the qualitative analysis of the constrained motions of the parts, represented in this place vocabulary, and the generation of the qualitative envisionment.

In all of these bodies of work a distinction is made between behavior and function. Behavior is represented as an envisionment, and is supposed to be as complete as possible. Function is not represented or inferred at all, it is knowledge external to the qualitative physical reasoning process—although in principle it could be brought to bear, for instance to determine if a given design can in fact achieve the function for which it is intended. Archytas attempts to directly infer both behavior and function from structure, but it does not attempt a qualitative envisionment; rather, it simply transfers behaviors of individual components and quantities from source to target following the structural mapping. That is, Archytas *uses* the existing structural analogy to infer behavior analogically, rather than attempting to derive a behavior from causal constraints expressed in a structural model (using confluences, qualitative processes, or qualitative differential equations).

Furthermore, the goal of Archytas is to infer function, and therefore to infer a functional decomposition of a a device that bottoms out in the function(s) of each individual component, but such

knowledge is not represented in qualitative physical models at all, which have explicitly disavowed functional knowledge. The difference is really a difference of goals: it was never the goal of qualitative physical reasoning to infer teleology, just as it was never the goal of SBF to provide for the inference of a qualitative envisionment.

7.3 Other Related Work

7.3.1 Visual Analogy

Visual analogy is a subject, not a field, and even as such it is far from well explored. Nevertheless, scouring the literature one can turn up an interesting and diverse selection of work.

One of the most interesting systems in this line of work is McGraw and Hofstadter's LetterSpirit [61, 52]. It takes a stylized seed letter as input and outputs an entire font that has the same style (or "spirit": imagine a table, each row containing the alphabet in some style; then each column represents a letter, and each row a spirit, hence the name). The system understands "roles", such as the crossbar of an f or a t. It makes new fonts by determining some attributes such as "role traits" (e.g. crossbar suppressed), "motifs" (geometric shapes, such as a parallelogram, used over and over again), and "abstract rules" (e.g. no diagonal lines, only horizontal and vertical), and using these attributes builds an entire alphabet in some new style. According to their theory, the concept of an A is a *fluid concept* that can be adapted from one context to the next.

A very early analogy system was Evans' ANALOGY program [23], which solved geometric analogies of the A : B :: C : ?? sort, the kind one might see on intelligence tests (see figure 33). Given a multiple choice question of this sort, ANALOGY attempted to find a procedure for turning A into B, and then turned C into each of the (say) 5 answers, chosing whichever of those transformation procedures (represented using a graph-based representation) was closest to the original. It had a language of primitives (dots, lines, closed polygons, etc.), relationships (above, left-of, and so on), and transformations (rotate, reflect, expand, contract, add, delete). The system is also described in some detail by Winston in his book [84, chapter 2].

PAN [64] is an example of a more recent system that performs the same task using case-based reasoning, and also uses graph-based representations. Its representation and inference mechanisms seem to have more generality and robustness than Evans' early system. Both of these systems,

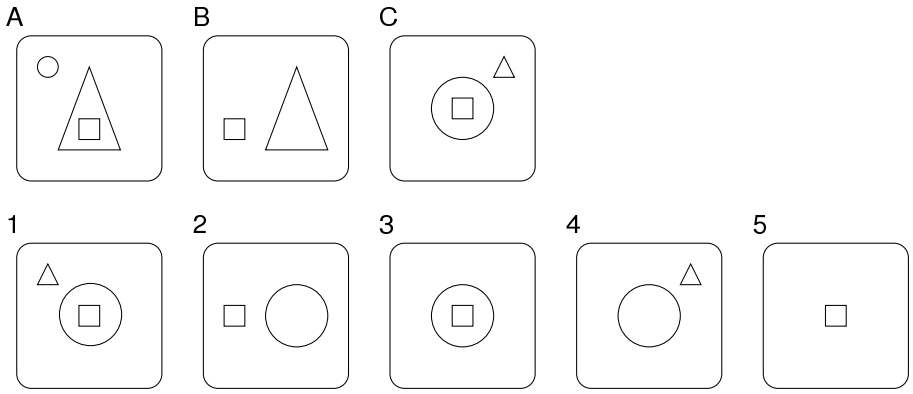


Figure 33: A geometric analogy problem of the sort seen on intelligence tests that is solved by Evans' ANALOGY program [23], of the form A : B :: C : ??, where the task is to choose one of 1, 2, 3, 4, or 5 for the remaining image.

ANALOGY as well as PAN, operate on what are essentially meaningless images in a very narrow task. It is more interesting to see what is required of a visual analogy system when the context of the reasoning task becomes broader and more meaningful, and the possible interpretations of the images in question become central.

The VAMP systems [78] of Thagard et al. are visual analogical mapping systems. VAMP.1 uses an array-based representation scheme, and maps source onto target by comparing symbols in the same location, scaling the arrays to be of the same size if they are different, converting both to the least common multiple of their respective sizes. The VAMP.2 system represents images as agents with local knowledge, which can be relational (e.g a is left of b), with each agent seeking to map itself to one in the target, and using ACME [53] to actually perform the mapping. VAMP.2 can find the correct mapping for the fortress/tumor problem of Gick and Holyoak [43] (that is, Duncker's radiation problem [21]) when it is represented diagrammatically. However, neither VAMP.1 nor VAMP.2 do anything like transfer and adaptation, and so cannot be said to be doing problem solving. At their core, they associate images with one another by parts. Transfer is required in order to solve the Duncker radiation problem.

the DIVA system [15] of Croft and Thagard is another analogical mapping system employing visual information. The system can represent scenes in 3-D using simple geometric shapes, and associate them with semantic networks intended to represent the content of these 3-D scenes. The system computes mappings again using ACME, and can find the mapping in one version of the

fortress/tumor problem. Once again, the system does not do transfer, and so is not actually performing any problem solving.

FABEL [36] was an early project employing case-based reasoning to explore the automated reuse of diagrammatic cases in the domain of architectural design, employing domain-specific heuristics to guide pattern extraction and transfer. FABEL is a large-scale project with many subsystems, designed as an expert system intended to demonstrate the practical usability of its various approaches to all the stages of the process, and it relies heavily on user interaction.

In chapter 2 I discussed the GeoRep system. GeoRep, once again, is a two-stage forward-chaining diagrammatic reasoner with a low-level domain-independent stage followed by a high-level domain-dependent stage. It can be used in a variety of different domains by making use of different rule sets in the high-level stage. GeoRep is part of a larger system called MAGI [27], which recognizes qualitative symmetry in diagrams, attempting to calculate the axis of symmetry for the figure (for instance, if the left and right halves of a figure were nearly symmetric, MAGI would draw a vertical line through the middle of the figure). MAGI uses the Structure-Mapping Engine (SME) [25] on the output of GeoRep, taking the base (or source) and target to be one and the representation of the diagram along with the added constraint to forbid the identity mapping. The JUXTA system [28] uses MAGI in its processing of so-called "juxtaposition diagrams", where some situation is represented in two diagrams side-by-side, and the alignable differences between them can be used to understand some situation. In principle, MAGI could be run on any sort of representation that SME could use, for example finding symmetry in a representation of a story.

The sort of symmetry with which MAGI is concerned is symmetry in the common sense notion of the word: a figure with a left and a right half, or a top and a bottom half, that are the same. The observation is that many figures are approximately symmetric but not exactly, so when asked people will not say they are symmetric, but they can easily find an axis of symmetry around which the figure can be flipped or rotated.

This is different from the usage of the word "symmetry" in this work. I employ the word in its technical mathematical sense, and furthermore the task is different. Archytas does not attempt to recognize symmetric figures, but simply to recognize when a given pattern matches a drawing. It does so by calculating what are called the symmetries of a shape. In mathematics, when a shape

maps onto itself (e.g. through an automorphism of the real plane) this mapping is called a symmetry of that shape, and the set of all such ways under the operation of functional composition (that is, $h(x) = g(f(x))$ is the composition $h = f \circ g$) is called a symmetry group of that shape. For instance, a square can be rotated about its center by 90°, 180° or 270° without altering its shape, and these are called *symmetries of a square*, and there are 8 of them in total, forming the so-called *dihedral group* D_4. I do not make use of symmetry *groups*, however, in this work, nor does Archytas attempt to calculate any important aspect of them for the shapes it reasons with.

7.3.2 Analogical and Case-Based Reasoning

The PRODIGY system [11, 82, 83] implements derivational analogy. In derivational analogy, transfer of problem-solving procedures from a source to a target uses memories of the justifications for each step, allowing for adaptation of the transferred procedure. A *derivation* is a trace, that is, a script of a problem solving procedure with justifications for taking of each particular step over others. Intermediate states generated are not stored, but are created anew each time a new analogy is made. There is some similarity here to the way Archytas transfers behavior, the primary difference being that Archytas must explicitly represent and transfer states, whereas PRODIGY does not.

CHEF [49] is a case-based planning system that adapts cooking recipes to solve new problems. Like PRODIGY, CHEF stores the sequence of operators used to derive the solution, but not intermediate states. Two other case-based reasoning systems are ARCHIE [65] and AskJef [4], which contain multi-modal cases, i.e. cases that contain both visual (e.g. photographs, drawings, diagrams, animations, and videos) and non-visual knowledge (e.g. goals, constraints, plans, and lessons). Nevertheless, the multi-modal cases in these systems typically are indexed only by non-visual constructs such as goals and constraints. The target problem too typically is specified by its goal and constraints, and cases are retrieved based on a match between the goals and constraints with the source cases. Like FABEL they are intended for use by people, and they rely on user interaction for the transfer step.

Holyoak and Thagard [54] developed the Process of Induction (PI) model, which uses a production system to solve problems, using spreading activation by activating concepts associated with rules as a side effect of rule firing, which can happen in parallel. This search for a problem-solving

procedure is bi-directional, employing both forward and backward chaining. The activation trace left from the activation of the source procedure guides it in a search for a new solution, which it then uses to generate an abstract schema that works as a single rule to apply to both problems in the future. This is not transfer in the usual sense of the word; instead, this model regards analogy as happening in three steps: retrieval, adaptation, and schema formation.

The Structure-Mapping Engine (SME) [25] was written as an implementation of Gentner's structure-mapping theory [37]. SME searches for mappings between source and target descriptions by regarding their representations as graph representations, and mapping the structures one-to-one onto each other, suggesting the transfer of particular relations from the source onto the target (so-called *candidate inferences*). SME treats analogical understanding as a mapping problem, where the goal of transfer is to make candidate inferences. This is different from model construction, which requires a different view of both mapping and transfer, as compositional analogy has shown, so that mapping and transfer must take place at different levels of abstraction. In addition to this, while SME is capable of generating multiple mappings for a given source and target, *an* analogy is *a particular* maping, whereas in Archytas, in the first stage, the shape analogy requires computing all mappings of a set of patterns and grouping them. Finally, SME uses a complex algorithm to compute partial mappings according to some heuristics, but Archytas must have exact matches for each pattern, as we have seen, and so the algorithm is much more straightforward.

PHINEAS [24] uses SME to make analogies between physical phenomena represented using Qualitative Process Theory, which is discussed above. It explains new behaviors by analogy to old behaviors, transferring knowledge from source to target. Evaluation takes place by qualitative simulation (in QPT), using past reasoning traces summarized by storing with each state an observation of the collection of theories used to explain it (e.g. with the example of alcohol evaporating from a flask, it may store theories about evaporation and containment). It can also hypothesize *skolem objects* to generate alignment with unmapped entities in the source analog. PHINEAS does not construct new models from scratch but, rather, attempts to understand physical analogies involving models that have already been supplied. PHINEAS does reason both at the level of structure and behavior, but it certainly does not operate at the level of drawing or shape, nor does it represent function.

The Analogical Constraint Mapping Engine (ACME) of Holyoak and Thagard [53], along with its complementary retrieval model Analogical Retrieval by Constraint Satisfaction (ARCS) [79], is a general analogy model. Holyoak and Thagard proposed that the retrieval and mapping tasks can be productively viewed as constraint satisfaction problems. Their proposal incorporated structural, semantic and pragmatic constraints and used graph isomorphism as the primary similarity measure. ACME and ARCS provided a "localist" connectionist implementation of their proposal. Nodes are constructed for each map hypothesis (between a source element and a target element), with inhibitory and excitatory links between the nodes, and the network is run until it reaches quiescence. A drawback is that all of the potential object or entity maps (that is, from each element in the source to each element in the target) must be "wired" into their network, which in even moderate-sized examples can grow to be quite large.

CHAPTER VIII

CONCLUSION

8.1 Claims

Before stating my conclusions, I review the hypotheses. The primary hypotheses of this work were as follows:

- *Method Hypothesis*: in order to analogically infer the teleology of a device depicted in a drawing, a reasoner must perform analogical inference at several levels of abstraction, using the inferences at one level as a basis for those at the next level, from shape through structure, causality, and teleology

- *Knowledge Hypothesis*: in order to represent the teleology of a device depicted in a drawing, a reasoner must capture knowledge of the shapes present in the drawing, the structural elements they depict, the causal processes of these elements, and the function of the device

In addition I added three hypotheses regarding the various levels of abstraction and the required stages of the reasoning process:

- *Dependency of Structural Knowledge on Shape*: Given a correct representation of the shapes in a target drawing, the correct structure may be inferred

- *Dependency of Causal Knowledge on Structural*: Given a correct representation of the structure of the device depicted in a drawing, a correct causal processes may be inferred

- *Dependency of Teleology on Causal Knowledge*: Given a correct representation of the causal processes of the device depicted in a drawing, a correct representation of the teleology may be inferred.

To address these hypotheses I devised my method of compositional analogy, implemented in and evaluated with the Archytas system.

In conclusion, the observations made in chapter 6 support my hypotheses:

> **Claim:** A correct representation of the function of a device depicted in an input drawing can only be inferred with a correct representation of causal behavior, structural elements, and shapes in the drawing

That is to say, the levels of abstraction are dependent on each other. To get to the level of function correctly required correct inferences at all of the previous levels. This follows from the nature of the representation, but also the nature of the problem.

Secondly:

> **Claim:** In order to analogically infer the teleology of a device depicted in a drawing, a reasoner must perform analogical inference at several levels of abstraction, using the inferences at one level as a bases for those at the next level, from shape through structure, causality, and teleology

This is a claim about the method required, that analogical inference at several levels of abstraction is required to solve the problem in question.

Finally:

> **Claim:** in order to represent the teleology of a device depicted in a drawing, a reasoner must capture knowledge of the shapes present in the drawing, the structural elements they depict, the causal processes of these elements, and the teleology of the device

That is to say, the knowledge must be knowledge captured in each case must be knowledge at all these same levels of abstraction.

The most significant contribution of this work is the creation of a method of analogical reasoning that can deal simultaneously with many levels of abstraction, starting from the very small and detailed and particular level of line segments and intersection points in a line drawing, and going all the way up to the highly abstract level of the function of the device depicted in that same drawing. This work has shown not only that it is possible to reason at all these levels, but that the task cannot be completed without reasoning at all these levels, and that furthermore the task may be done using analogical reasoning. The use of an analogical reasoning process in which, for instance, a whole mapping of a network of relations can become a single map at the next level of abstraction, or how

layers of mappings can be iterated over and used for the transfer of model elements at the next level, is entirely novel. It is the bringing together of drawings, of knowledge of shape, structure, causality, and teleology that makes this possible.

8.2 Future Work

There are several places in this work that have suggested opportunities for future investigation.

The shape matching algorithm appears to be quite fast, even though subgraph isomorphism is known to be NP-Complete in general. However, David Eppstein has shown [22] that for a fixed pattern, a planar graph can be found to be a subgraph of another planar graph in *linear* time in the complexity of the target. My intersection graphs are not planar graphs in general, but this suggests it may be possible to show bounds on the complexity of the algorithm based on the properties of the representation. This problem is likely to be a complex one, however.

Along similar theoretical lines, each particular composite shape had a fixed number of mappings in each mapping group across different targets. This is interesting observation, especially in light of the brief discussion of symmetry groups in §7.3.1. This may be a key to further theoretical analysis of the shape mapping method and its limitations.

The interaction between shape mappings and the component model are complex, and suggest multiple interacting constraints. It was observed that a kind of symbolic mismatch occurred, where two different components had the same shape which then matched the same pattern in the target, but two components were transferred. The complexities added by conflicting shape models have not been adequately considered, and may require a more complex algorithm. For instance, can different components be of the same kind, and therefore have the same shape representation? If so, then the cleanup phase of the structural model transfer step, where disconnected chains get removed, must get much more complex. In addition, as was observed, Archytas's method is sensitive to order: if component A is removed before component B, it may delete only the one component and not the whole chain, but if it happens to be ordered in memory the other way around it may delete the entire model. This suggests that determining which shapes and structural elements to keep or reject in some interpretation of a drawing is really a constraint satisfaction problem, in which case some form of symbolic relaxation, say, might produce far better results in many cases.

As it is, Archytas merely transfers behaviors and assumes it can hook them up as according to the sphere of influence of the relevant component. However, in some cases it may be necessary to recompute the behavior of some component if the model has changed sufficiently. In particular, we saw that when there was a reversal in the direction of causality, the transfer method broke down. Solving this requires a more complex theory of the transfer and adaptation of behavior and function than has been explored in this work, something which may involve both qualitative physical reasoning as well as analogical transfer.

More generally, the theory of compositional analogy has been one of vertical integration at several different levels of abstraction and aggregation. But at each level, one can imagine a flattening, a increase in the reasoning power of the system *at that level* to deal with more kinds of domains. Can the representation of shapes be made more rich and more robust? Can the matching of shapes be made to be configurable, so that different constraints can be applied when needed, but not otherwise, depending say on the domain or even the particular case? Is it possible for the structural transfer make use of functional decomposition to determine if all of the required roles are being filled, and thus transfer additional elements or remove some spurious elements? Might the behavioral transfer be generalized so as to be capable of some (perhaps limited) qualitative physical inference? Might the representation of function be made to be more expressive so that the system can capture the function of unusual devices? All these questions involve extending the range of some particular module within any implementation of this theory.

Finally, the most fruitful avenue of exploration with this work will no doubt be further application domains. In particular, I would expect aesthetic design, which has a more explicitly analogical nature to it, to be a better domain on which to attempt methods such as this. Another potentially fruitful avenue of exploration is that of biologically inspired design, again with a more explicitly analogical nature to it. In both cases the question is whether these methods can correctly interpret the analogies being made by designers at a visual level, and so if this can again be an effective means for knowledge acquisition in design. I believe this to be a promising direction for future work.

APPENDIX A

SOURCE MODELS FOR ARCHYTAS

This appendix contains the complete specification of the five source models described in chapter 3, and used in chapter 6, shown in the input format. Each source model is specified in a model file that refers to the drawing file and also contains specifications of the locations of the shapes for each component and connection.

A.1 Piston and Crankshaft Model

This model was the source model for examples **PC1**, **PC2**, and **PC3** (recall the key in table 26, page 93). The structural model was described in §3.2.1 (page 44), the behavior and function in §3.3.3 (page 53).

The declaration of the model:

```
(def-model piston-crankshaft "figs/devicefigs/piston-diagram.fig")
```

The drawing referred to is figure 1(a) (page 2). The quantities are specified as follows:

```
(def-quantity piston-motion (piston-crankshaft)
  :parameters
  ((piston-velocity vector (downward upward))
   (piston-position vector (cyl-bot cyl-top))))

(def-quantity rod-linear-motion (piston-crankshaft)
  :parameters
  ((rod-linear-velocity vector (downward upward))
   (rod-position vector (cyl-bot cyl-mid cyl-top))))

(def-quantity rod-angular-motion (piston-crankshaft)
  :parameters
```

```
  ((piston-rod-angular-velocity vector (clockwise 0 counterclockwise))
   (piston-rod-angle angle (rho- 0 rho+))
   (crank-rod-angular-velocity vector (clockwise 0))
   (crank-rod-angle angle (phi- 0 phi+ pi))))

(def-quantity crank-motion (piston-crankshaft)
  :parameters
  ((crank-velocity vector (0 counterclockwise))
   (crank-angle angle (0 pi))))
```

The model components are:

```
(def-component piston (piston-crankshaft)
  :primitive-functions
  ((allow piston-motion))
  :shape
  (piston-rect :isecs ((point 2850.0 6000.0)
                       (point 2850.0 4800.0)
                       (point 2250.0 6000.0)
                       (point 2250.0 4800.0))
               :circles (((point 2625.0 5400.0) 150.0))))

(def-component cylinder (piston-crankshaft)
  :properties
  ((cyl-bot vector cylinder-bottom-position)
   (cyl-mid vector cylinder-middle-position)
   (cyl-top vector cylinder-top-position))
  :shape
  (cyl-rects :isecs ((point  900.0 6000.0)
                     (point 3825.0 6000.0)
```

```
                    (point  900.0 6600.0)
                    (point 3825.0 6600.0)
                    (point  900.0 4200.0)
                    (point 3825.0 4200.0)
                    (point  900.0 4800.0)
                    (point 3825.0 4800.0))
             :minus piston-rect))

(def-component crankcase (piston-crankshaft))

(def-component connecting-rod (piston-crankshaft)
  :primitive-functions
  ((allow rod-linear-motion)
   (allow rod-angular-motion))
  :shape
  (rod-rect :isecs ((point 6643.0 4559.0)
                    (point 6527.0 4124.0)
                    (point 2441.0 5685.0)
                    (point 2325.0 5250.0))
            :circles (((point 2625.0 5400.0) 150.0)
                      ((point 6300.0 4425.0) 150.0))))

(def-component crankshaft (piston-crankshaft)
  :primitive-functions
  ((allow crank-motion))
  :shape
  (crank-cir :circles (((point 6450.0 5400.0) 225.0))
             :arcs (((point 6450.0 5400.0) 1350.0))))
```

The structural relations between components (i.e. connections):

```
(def-connection piston-crankshaft
    (cylindrical-joint piston cylinder)
  :composite-shape
  (piston-cyl-shape (piston-rect cyl-rects)))

(def-connection piston-crankshaft
    (revolute-joint piston connecting-rod)
  :composite-shape
  (piston-rod-shape (piston-rect rod-rect)))

(def-connection piston-crankshaft
    (revolute-joint crankshaft connecting-rod)
  :composite-shape
  (crank-rod-shape (crank-cir rod-rect)))

(def-connection piston-crankshaft
    (revolute-joint crankshaft crankcase))

(def-connection piston-crankshaft
    (fused cylinder crankcase))
```

The structural relations between components and quantities (quantity relations):

```
(def-quantity-relation piston-crankshaft
    (contains piston piston-motion))

(def-quantity-relation piston-crankshaft
    (contains connecting-rod rod-linear-motion))

(def-quantity-relation piston-crankshaft
```

```
    (contains connecting-rod rod-angular-motion))


(def-quantity-relation piston-crankshaft
    (contains crankshaft crank-motion))
```

This completes the structural model.

The behavioral model has four behaviors. First, the piston motion:

```
(def-quantity-behavior piston-crankshaft
    (piston-motion piston-velocity
                   piston-position)
  :state
  (p-s1 (piston-position cyl-top)
        (piston-velocity downward))
  :transition
  (trans (p-s1 p-s2)
         :using-function
         ((allow piston piston-motion))
         :under-condition-structure
         ((cylindrical-joint piston cylinder))
         :due-to-stimulus
         (piston-downforce pf))
  :state
  (p-s2 (piston-position cyl-bot)
        (piston-velocity upward))
  :transition
  (trans (p-s2 p-s1)
         :using-function
         ((allow piston piston-motion))
         :under-condition-structure
```

```
        ((cylindrical-joint piston cylinder)
         (revolute-joint piston connecting-rod))
        :under-condition-transition
        ((rod-linear-motion rl-s2 rl-s1))))
```

The linear motion of the connecting rod:

```
(def-quantity-behavior piston-crankshaft
    (rod-linear-motion rod-linear-velocity
                       rod-position)
  :state
  (rl-s1 (rod-position cyl-top)
         (rod-linear-velocity downward))
  :transition
  (trans (rl-s1 rl-s2)
         :using-function
         ((allow connecting-rod rod-linear-motion))
         :under-condition-structure
         ((revolute-joint piston connecting-rod))
         :under-condition-transition
         ((piston-motion p-s1 p-s2))
         :parametric-equation
         (= (piston-position piston-motion)
            (rod-position rod-linear-motion)))
  :state
  (rl-s2 (rod-position cyl-bot)
         (rod-linear-velocity upward))
  :transition
  (trans (rl-s2 rl-s1)
         :using-function
```

```
        ((allow connecting-rod rod-linear-motion))
        :under-condition-structure
        ((revolute-joint crankshaft connecting-rod))
        :under-condition-transition
        ((rod-angular-motion ra-s3 ra-s4)
         (rod-angular-motion ra-s4 ra-s1))
        :parametric-equation
        (= (piston-position piston-motion)
           (rod-position rod-linear-motion))))
```

The rotational (or "angular") motion of the connecting rod:

```
(def-quantity-behavior piston-crankshaft
    (rod-angular-motion piston-rod-angular-velocity
                        piston-rod-angle
                        crank-rod-angular-velocity
                        crank-rod-angle)
  :state
  (ra-s1 (piston-rod-angle 0)
         (piston-rod-angular-velocity clockwise)
         (crank-rod-angle pi)
         (crank-rod-angular-velocity clockwise))
  :transition
  (trans (ra-s1 ra-s2)
         :using-function
         ((allow connecting-rod rod-angular-motion))
         :under-condition-structure
         ((revolute-joint piston connecting-rod)
          (revolute-joint crankshaft connecting-rod))
         :under-condition-transition
```

```
       ((rod-linear-motion rl-s1 rl-s2)))
:state
(ra-s2 (piston-rod-angle rho-)
       (piston-rod-angular-velocity counterclockwise)
       (crank-rod-angle phi+)
       (crank-rod-angular-velocity clockwise))
:transition
(trans (ra-s2 ra-s3)
       :using-function
       ((allow connecting-rod rod-angular-motion))
       :under-condition-structure
       ((revolute-joint piston connecting-rod)
        (revolute-joint crankshaft connecting-rod))
       :under-condition-transition
       ((rod-linear-motion rl-s1 rl-s2)))
:state
(ra-s3 (piston-rod-angle 0)
       (piston-rod-angular-velocity counterclockwise)
       (crank-rod-angle 0)
       (crank-rod-angular-velocity clockwise))
:transition
(trans (ra-s3 ra-s4)
       :using-function
       ((allow connecting-rod rod-angular-motion))
       :under-condition-structure
       ((revolute-joint piston connecting-rod)
        (revolute-joint crankshaft connecting-rod))
       :under-condition-transition
       ((rod-linear-motion rl-s2 rl-s1)))
```

```
  :state
  (ra-s4 (piston-rod-angle rho+)
         (piston-rod-angular-velocity clockwise)
         (crank-rod-angle phi-)
         (crank-rod-angular-velocity clockwise))
  :transition
  (trans (ra-s4 ra-s1)
         :using-function
         ((allow connecting-rod rod-angular-motion))
         :under-condition-structure
         ((revolute-joint piston connecting-rod)
          (revolute-joint crankshaft connecting-rod))
         :under-condition-transition
         ((rod-linear-motion rl-s2 rl-s1))))
```

And finally, the crankshaft motion:

```
(def-quantity-behavior piston-crankshaft
    (crank-motion crank-velocity
                  crank-angle)
  :state
  (c-s1 (crank-angle 0)
        (crank-velocity counterclockwise))
  :transition
  (trans (c-s1 c-s2)
         :using-function
         ((allow crankshaft crank-motion))
         :under-condition-structure
         ((revolute-joint crankshaft crankcase)
          (revolute-joint crankshaft connecting-rod))
```

```
        :under-condition-transition
        ((rod-angular-motion ra-s1 ra-s2)
         (rod-angular-motion ra-s2 ra-s3))
        :under-condition-state
        ((rod-angular-motion ra-s2)))
  :state
  (c-s2 (crank-angle pi)
        (crank-velocity counterclockwise))
  :transition
  (trans (c-s2 c-s1)
        :using-function
        ((allow crankshaft crank-motion))
        :under-condition-structure
        ((revolute-joint crankshaft crankcase)
         (revolute-joint crankshaft connecting-rod))
        :as-per
        ((principle rotational-intertia
                    (> (crank-velocity crank-motion) 0))
         (principle friction
                    (< (crank-velocity crank-motion c-s2)
                       (crank-velocity crank-motion c-s1))))))
```

These are the behaviors of this model.

The functional specification is as follows:

```
(def-functional-spec piston-crankshaft turn-crankshaft
  :given (crank-motion c-s1)
  :makes (crank-motion c-s2)
  :by-behavior (behavior crank-motion))
```

A.2 Door Latch Model

This model was the source model for examples **DL1** and **DL2**.

First, the model declaration:

```
(def-model doorlatch "figs/devicefigs/door-latch-diagram-d.fig")
```

The drawing here is figure 20 (page 46).

Next is the structural model. There are no quantities in this model (and, hence, no primitive functions, either). And so:

```
(def-component door (doorlatch)
  :properties
  ((in vector bolt-in-position)
   (out vector bolt-out-position)
   (theta angle cam-open-angle))
  :shape
  (door-rects :isecs ((point 8100.0 10200.0)
                      (point 8400.0 10200.0)
                      (point 8400.0  8250.0)
                      (point 8100.0  8250.0)
                      (point 8100.0  7350.0)
                      (point 8400.0  7350.0)
                      (point 8400.0  5400.0)
                      (point 8100.0  5400.0)
                      (point 8250.0  5400.0)
                      (point 8250.0  4500.0)
                      (point 1800.0  4500.0)
                      (point 1800.0 11100.0)
                      (point 8250.0 11100.0)
                      (point 8250.0 10200.0))))
```

```
(def-component cam (doorlatch)
  :parameters
  ((cam-angle angle (0 theta)))
  :shape
  (cam-cir :isecs ((point 4161.00 8918.00)
                   (point 4271.10 8927.57)
                   (point 4460.00 8944.00)
                   (point 4473.94 8785.00)
                   (point 4591.00 7450.00)
                   (point 4291.00 7424.00)
                   (point 4211.34 8339.50)
                   (point 4183.41 8660.50))))

(def-component shaft (doorlatch)
  :parameters
  ((shaft-position vector (in out)))
  :shape
  (shaft-shape :isecs ((point 3900.0 7950.0)
                       (point 7500.0 7950.0)
                       (point 7500.0 7875.0)
                       (point 7500.0 7725.0)
                       (point 5100.0 7725.0)
                       (point 5100.0 7875.0)
                       (point 7500.0 7650.0)
                       (point 3900.0 7650.0)
                       (point 4050.0 7950.0)
                       (point 4050.0 7650.0)
                       (point 3900.0 7800.0))
               :arcs (((point 4050.0 7800.0) 150.0))))
```

```
(def-component bolt (doorlatch)
  :parameters
  ((bolt-position vector (in out)))
  :shape
  (bolt-rect :isecs ((point 7500.0 8250.0)
                     (point 9300.0 8250.0)
                     (point 9300.0 7350.0)
                     (point 7500.0 7350.0))))
```

Connections between components:

```
(def-connection doorlatch
    (revolute-joint cam door)
  :composite-shape
  (door-cam-shape (door-rects cam-cir)))

(def-connection doorlatch
    (fused shaft bolt)
  :composite-shape
  (shaft-bolt-shape (shaft-shape bolt-rect)))

(def-connection doorlatch
    (prismatic-joint bolt door)
  :composite-shape
  (door-bolt-shape (door-rects bolt-rect)))

(def-connection doorlatch
    (hook-joint cam shaft)
  :composite-shape
```

```
  (cam-shaft-shape (cam-cir shaft-shape)))
```

This completes the structural model.

The behavioral model has three behaviors. First, the behavior of the cam:

```
(def-component-behavior doorlatch
    (cam cam-angle)
  :state
  (c-s1 (cam-angle 0))
  :transition
  (trans (c-s1 c-s2)
         :under-condition-structure
         ((revolute-joint cam door))
         :due-to-stimulus
         (turn-handle th))
  :state
  (c-s2 (cam-angle theta)))
```

The shaft:

```
(def-component-behavior doorlatch
    (shaft shaft-position)
  :state
  (sh-s1 (shaft-position out))
  :transition
  (trans (sh-s1 sh-s2)
         :under-condition-structure
         ((hook-joint cam shaft))
         :under-condition-transition
         ((cam c-s1 c-s2)))
  :state
  (sh-s2 (shaft-position in)))
```

And, finally, the bolt:

```
(def-component-behavior doorlatch
    (bolt bolt-position)
  :state
  (b-s1 (bolt-position out))
  :transition
  (trans (b-s1 b-s2)
         :under-condition-structure
         ((fused shaft bolt)
          (prismatic-joint bolt door))
         :under-condition-transition
         ((shaft sh-s1 sh-s2)))
  :state
  (b-s2 (bolt-position in)))
```

Thus completing the behavioral model.

The functional specification:

```
(def-functional-spec doorlatch
    move-bolt
  :given (bolt b-s1)
  :makes (bolt b-s2)
  :by-behavior (behavior bolt))
```

Thus completing the model.

A.3 Pulley Input Models

A.3.1 Original Pulley Model

This was the source model for example **PL1**. Model declaration:

```
(def-model pulley "figs/devicefigs/pulley1.fig")
```

The drawing is shown in figure 23 (page 49).

Starting with the structural model, first the quantities:

```
(def-quantity wheel-a-angular-motion (pulley)
  :parameters
  ((wheel-a-velocity vector (0 clockwise))))

(def-quantity wheel-b-angular-motion (pulley)
  :parameters
  ((wheel-b-angular-velocity vector (0 counterclockwise))))

(def-quantity wheel-b-linear-motion (pulley)
  :parameters
  ((wheel-b-linear-velocity vector (0 upwards))))

(def-quantity rope-a-motion (pulley)
  :parameters
  ((rope-a-velocity vector (0 downwards))))

(def-quantity rope-b-motion (pulley)
  :parameters
  ((rope-b-velocity vector (0 upwards))))

(def-quantity fixed-rope-b-motion (pulley)
  :parameters
  ((fixed-rope-b-velocity vector (0 upwards))))

(def-quantity weight-motion (pulley)
  :parameters
  ((weight-velocity vector (0 upwards))))
```

Next, the components:

```
(def-component wheel-a (pulley)
  :primitive-functions
  ((allow wheel-a-angular-motion))
  :shape
  (wheel-a-cir :isecs ((point 2700.0 9000.0)
                       (point 3075.0 9290.47)
                       (point 2925.0 9290.47)
                       (point 3300.0 9000.0)
                       (point 2925.0 9600.0)
                       (point 2925.0 8925.0)
                       (point 3075.0 8925.0)
                       (point 3075.0 9600.0))))

(def-component wheel-b (pulley)
  :primitive-functions
  ((allow wheel-b-angular-motion)
   (allow wheel-b-linear-motion))
  :shape
  (wheel-b-cir :isecs ((point 2100.0 6600.0)
                       (point 2475.0 6309.53)
                       (point 2325.0 6309.53)
                       (point 2700.0 6600.0)
                       (point 2325.0 6675.0)
                       (point 2325.0 6000.0)
                       (point 2475.0 6000.0)
                       (point 2475.0 6675.0))))

(def-component ceiling (pulley)
```

```
  :shape
  (ceiling-rect :isecs ((point 1200.0 10800.0)
                        (point 1200.0 10200.0)
                        (point 4200.0 10200.0)
                        (point 4200.0 10800.0))))

(def-component fixed-rope-a (pulley)
  :shape
  (fixed-rope-a-line :isecs ((point 3000.0  9600.0)
                             (point 3000.0 10200.0))))

(def-component fixed-rope-b (pulley)
  :primitive-functions
  ((allow fixed-rope-b-motion))
  :shape
  (fixed-rope-b-line :isecs ((point 2400.0 6000.0)
                             (point 2400.0 5325.0))))

(def-component rope-a (pulley)
  :primitive-functions
  ((allow rope-a-motion))
  :shape
  (rope-a-line :isecs ((point 3300.0 9000.0)
                       (point 3300.0 7800.0)
                       (point 3375.0 7725.0)
                       (point 3225.0 7725.0))))

(def-component rope-b (pulley)
  :primitive-functions
```

```
  ((allow rope-b-motion))
  :shape
  (rope-b-line :isecs ((point 2700.0 9000.0)
                       (point 2700.0 6600.0))))

(def-component rope-c (pulley)
  :shape
  (rope-c-line :isecs ((point 2100.0  6600.0)
                       (point 2100.0 10200.0))))

(def-component weight (pulley)
  :properties
  ((mass scalar mass-val))
  :primitive-functions
  ((allow weight-motion))
  :shape
  (weight-rect :isecs ((point 1800.0 5325.0)
                       (point 1800.0 4725.0)
                       (point 3000.0 4725.0)
                       (point 3000.0 5325.0))))
```

Next up, connections:

```
(def-connection pulley
    (rolling-joint rope-a wheel-a)
  :composite-shape
  (wheel-a-rope-a-shape (rope-a-line wheel-a-cir)))

(def-connection pulley
    (rolling-joint rope-b wheel-a)
```

```
  :composite-shape
  (wheel-a-rope-b-shape (rope-b-line wheel-a-cir)))

(def-connection pulley
    (rolling-joint rope-b wheel-b)
  :composite-shape
  (wheel-b-rope-b-shape (rope-b-line wheel-b-cir)))

(def-connection pulley
    (rolling-joint rope-c wheel-b)
  :composite-shape
  (wheel-b-rope-c-shape (rope-c-line wheel-b-cir)))

(def-connection pulley
    (attached wheel-a fixed-rope-a)
  :composite-shape
  (wheel-a-fixed-rope-a-shape (wheel-a-cir fixed-rope-a-line)))

(def-connection pulley
    (attached fixed-rope-a ceiling)
  :composite-shape
  (fixed-rope-a-ceiling-shape (fixed-rope-a-line ceiling-rect)))

(def-connection pulley
    (attached wheel-b fixed-rope-b)
  :composite-shape
  (wheel-b-fixed-rope-b-shape (wheel-b-cir fixed-rope-b-line)))

(def-connection pulley
```

```
    (attached fixed-rope-b weight)
  :composite-shape
  (fixed-rope-b-weight-shape (fixed-rope-b-line weight-rect)))

(def-connection pulley
    (attached rope-c ceiling)
  :composite-shape
  (rope-c-ceiling-shape (rope-c-line ceiling-rect)))
```

And finally, quantity relations:

```
(def-quantity-relation pulley
    (contains wheel-a wheel-a-angular-motion))

(def-quantity-relation pulley
    (contains wheel-b wheel-b-angular-motion))

(def-quantity-relation pulley
    (contains wheel-b wheel-b-linear-motion))

(def-quantity-relation pulley
    (contains rope-a rope-a-motion))

(def-quantity-relation pulley
    (contains rope-b rope-b-motion))

(def-quantity-relation pulley
    (contains fixed-rope-b fixed-rope-b-motion))

(def-quantity-relation pulley
```

```
    (contains weight weight-motion))
```

Thus completing the structural model of the pulley.

There are seven behaviors in this model. First, the rope *A* motion:

```
(def-quantity-behavior pulley
    (rope-a-motion rope-a-velocity)
  :state
  (r-a-s1 (rope-a-velocity 0))
  :transition
  (trans (r-a-s1 r-a-s2)
         :using-function
         ((allow rope-a rope-a-motion))
         :due-to-stimulus
         (pull-rope p))
  :state
  (r-a-s2 (rope-a-velocity downwards)))
```

Next, wheel *A* motion:

```
(def-quantity-behavior pulley
    (wheel-a-angular-motion wheel-a-velocity)
  :state
  (w-a-s1 (wheel-a-velocity 0))
  :transition
  (trans (w-a-s1 w-a-s2)
         :using-function
         ((allow wheel-a wheel-a-angular-motion))
         :under-condition-structure
         ((rolling-joint rope-a wheel-a))
         :under-condition-transition
         ((rope-a-motion r-a-s1 r-a-s2)))
```

```
  :state
  (w-a-s2 (wheel-a-velocity clockwise)))
```

Rope *B* motion:

```
(def-quantity-behavior pulley
    (rope-b-motion rope-b-velocity)
  :state
  (r-b-s1 (rope-b-velocity 0))
  :transition
  (trans (r-b-s1 r-b-s2)
         :using-function
         ((allow rope-b rope-b-motion))
         :under-condition-structure
         ((rolling-joint rope-b wheel-a))
         :under-condition-transition
         ((wheel-a-angular-motion w-a-s1 w-a-s2)))
  :state
  (r-b-s2 (rope-b-velocity upwards)))
```

Wheel *B* rotational (="angular") motion:

```
(def-quantity-behavior pulley
    (wheel-b-angular-motion wheel-b-angular-velocity)
  :state
  (w-b-ang-s1 (wheel-b-angular-velocity 0))
  :transition
  (trans (w-b-ang-s1 w-b-ang-s2)
         :using-function
         ((allow wheel-b wheel-b-angular-motion))
         :under-condition-structure
         ((rolling-joint rope-b wheel-b)
```

```
          (rolling-joint rope-c wheel-b))
         :under-condition-transition
         ((rope-b-motion r-b-s1 r-b-s2)))
  :state
  (w-b-ang-s2 (wheel-b-angular-velocity counterclockwise)))
```

Wheel *B* linear motion:

```
(def-quantity-behavior pulley
    (wheel-b-linear-motion wheel-b-linear-velocity)
  :state
  (w-b-ln-s1 (wheel-b-linear-velocity 0))
  :transition
  (trans (w-b-ln-s1 w-b-ln-s2)
         :using-function
         ((allow wheel-b wheel-b-linear-motion))
         :under-condition-structure
         ((rolling-joint rope-b wheel-b)
          (rolling-joint rope-c wheel-b))
         :under-condition-transition
         ((wheel-b-angular-motion w-b-ang-s1 w-b-ang-s2)))
  :state
  (w-b-ln-s2 (wheel-b-linear-velocity upwards)))
```

So-called "fixed" rope *B* motion (fixed in the sense that it holds something in place rather than moves through one of the pulley wheels):

```
(def-quantity-behavior pulley
    (fixed-rope-b-motion fixed-rope-b-velocity)
  :state
  (fr-b-s1 (fixed-rope-b-velocity 0))
  :transition
```

```
  (trans (fr-b-s1 fr-b-s2)
         :using-function
         ((allow fixed-rope-b fixed-rope-b-motion))
         :under-condition-structure
         ((attached wheel-b fixed-rope-b))
         :under-condition-transition
         ((fixed-rope-b-motion fr-b-s1 fr-b-s2)))
  :state
  (fr-b-s2 (fixed-rope-b-velocity upwards)))
```

And finally, the motion of the weight that the pulley lifts:

```
(def-quantity-behavior pulley
    (weight-motion weight-velocity)
  :state
  (w-s1 (weight-velocity 0))
  :transition
  (trans (w-s1 w-s2)
         :using-function
         ((allow weight weight-motion))
         :under-condition-structure
         ((attached fixed-rope-b weight))
         :under-condition-transition
         ((fixed-rope-b-motion fr-b-s1 fr-b-s2)))
  :state
  (w-s2 (weight-velocity upwards)))
```

This completes the behavioral model.

The functional specification then is:

```
(def-functional-spec pulley lift-weight
  :given (weight-motion w-s1)
```

```
  :makes (weight-motion w-s2)
  :by-behavior (behavior weight-motion))
```

And this completes the first model of the pulley.

A.3.2 Modified Pulley Model

This was the source model for example **PL1a**, **PL2a**, **PL3a**, and **PL4a**. This model is identical to the previous pulley model, the only difference is in the specification of the basic shapes. The drawing is shown in figure 28(a) (page 57). Basic shapes are given in the components they depict. As such, the revised components are:

```
(def-component wheel-a (pulley)
  :primitive-functions
  ((allow wheel-a-angular-motion))
  :shape
  (wheel-a-cir :isecs ((point 2700.0 9000.0)
                       (point 3075.0 9290.47)
                       (point 2925.0 9290.47)
                       (point 3300.0 9000.0)
                       (point 2925.0 9600.0)
                       (point 2925.0 9000.0)
                       (point 3000.0 8925.0)
                       (point 3075.0 9000.0)
                       (point 3075.0 9600.0))))


(def-component wheel-b (pulley)
  :primitive-functions
  ((allow wheel-b-angular-motion)
   (allow wheel-b-linear-motion))
  :shape
  (wheel-b-cir :isecs ((point 2100.0 6600.0)
```

```
                        (point 2475.0 6309.53)
                        (point 2325.0 6309.53)
                        (point 2700.0 6600.0)
                        (point 2325.0 6600.0)
                        (point 2325.0 6000.0)
                        (point 2475.0 6000.0)
                        (point 2475.0 6600.0)
                        (point 2400.0 6675.0))))

(def-component ceiling (pulley)
  :shape
  (ceiling-rect :isecs ((point 1200.0 10800.0)
                        (point 1200.0 10200.0)
                        (point 4200.0 10200.0)
                        (point 4200.0 10800.0))))

(def-component fixed-rope-a (pulley)
  :shape
  (fixed-rope-a-line :isecs ((point 3000.0  9600.0)
                             (point 3000.0 10200.0))))

(def-component fixed-rope-b (pulley)
  :primitive-functions
  ((allow fixed-rope-b-motion))
  :shape
  (fixed-rope-b-line :isecs ((point 2400.0 6000.0)
                             (point 2400.0 5550.0))))

(def-component rope-a (pulley)
```

```
  :primitive-functions
  ((allow rope-a-motion))
  :shape
  (rope-a-line :isecs ((point 3300.0 9000.0)
                       (point 3300.0 7800.0)
                       (point 3375.0 7725.0)
                       (point 3225.0 7725.0))))

(def-component rope-b (pulley)
  :primitive-functions
  ((allow rope-b-motion))
  :shape
  (rope-b-line :isecs ((point 2700.0 9000.0)
                       (point 2700.0 6600.0))))

(def-component rope-c (pulley)
  :shape
  (rope-c-line :isecs ((point 2100.0  6600.0)
                       (point 2100.0 10200.0))))

(def-component weight (pulley)
  :properties
  ((mass scalar mass-val))
  :primitive-functions
  ((allow weight-motion))
  :shape
  (weight-rect :isecs ((point 1800.0 5325.0)
                       (point 1800.0 4725.0)
                       (point 3000.0 4725.0)
```

```
                        (point 3000.0 5325.0)
                        (point 2400.0 5550.0))))
```

The remainder of the model is identical to that in §A.3.1.

A.3.3 Simplified Pulley Model

This was the source model for example **PL1b**.

The model declaration:

```
(def-model pulley "figs/devicefigs/pulley1b.fig")
```

The drawing is shown in figure 29(a) (page 58).

The structural model begins with these quantities:

```
(def-quantity wheel-angular-motion (pulley)
  :parameters
  ((wheel-velocity vector (0 clockwise))))

(def-quantity rope-a-motion (pulley)
  :parameters
  ((rope-a-velocity vector (0 downwards))))

(def-quantity rope-b-motion (pulley)
  :parameters
  ((rope-b-velocity vector (0 upwards))))

(def-quantity weight-motion (pulley)
  :parameters
  ((weight-velocity vector (0 upwards))))
```

The components:

```
(def-component wheel (pulley)
```

```
  :primitive-functions
  ((allow wheel-angular-motion))
  :shape
  (wheel-cir :isecs ((point 2700.0 9000.0)
                     (point 3075.0 9290.47)
                     (point 2925.0 9290.47)
                     (point 3300.0 9000.0)
                     (point 2925.0 9600.0)
                     (point 2925.0 9000.0)
                     (point 3000.0 8925.0)
                     (point 3075.0 9000.0)
                     (point 3075.0 9600.0))))

(def-component ceiling (pulley)
  :shape
  (ceiling-rect :isecs ((point 1800.0 10800.0)
                        (point 1800.0 10200.0)
                        (point 4200.0 10200.0)
                        (point 4200.0 10800.0))))

(def-component fixed-rope (pulley)
  :shape
  (fixed-rope-line :isecs ((point 3000.0  9600.0)
                           (point 3000.0 10200.0))))

(def-component rope-a (pulley)
  :primitive-functions
  ((allow rope-a-motion))
  :shape
```

```
  (rope-a-line :isecs ((point 3300.0 9000.0)
                       (point 4200.0 7200.0)
                       (point 4125.0 7125.0)
                       (point 4275.0 7125.0))))

(def-component rope-b (pulley)
  :primitive-functions
  ((allow rope-b-motion))
  :shape
  (rope-b-line :isecs ((point 2700.0 9000.0)
                       (point 2700.0 6825.0))))

(def-component weight (pulley)
  :properties
  ((mass scalar mass-val))
  :primitive-functions
  ((allow weight-motion))
  :shape
  (weight-rect :isecs ((point 2100.0 6600.0)
                       (point 2100.0 6000.0)
                       (point 3300.0 6000.0)
                       (point 3300.0 6600.0)
                       (point 2700.0 6825.0))))
```

Connections:

```
(def-connection pulley
    (rolling-joint rope-a wheel)
  :composite-shape
  (wheel-rope-a-shape (rope-a-line wheel-cir)))
```

```
(def-connection pulley
    (rolling-joint rope-b wheel)
  :composite-shape
  (wheel-rope-b-shape (rope-b-line wheel-cir)))

(def-connection pulley
    (attached wheel fixed-rope)
  :composite-shape
  (wheel-fixed-rope-shape (wheel-cir fixed-rope-line)))

(def-connection pulley
    (attached fixed-rope ceiling)
  :composite-shape
  (fixed-rope-ceiling-shape (fixed-rope-line ceiling-rect)))

(def-connection pulley
    (attached rope-b weight)
  :composite-shape
  (rope-b-weight-shape (rope-b-line weight-rect)))
```

And the quantity relations:

```
(def-quantity-relation pulley
    (contains wheel wheel-angular-motion))

(def-quantity-relation pulley
    (contains rope-a rope-a-motion))

(def-quantity-relation pulley
```

```
    (contains rope-b rope-b-motion))


(def-quantity-relation pulley
    (contains weight weight-motion))
```

Thus completing the structural model.

The behavioral model is simpler, with only four behaviors. First, rope *A* motion:

```
(def-quantity-behavior pulley
    (rope-a-motion rope-a-velocity)
  :state
  (r-a-s1 (rope-a-velocity 0))
  :transition
  (trans (r-a-s1 r-a-s2)
         :using-function
         ((allow rope-a rope-a-motion))
         :due-to-stimulus
         (pull-rope p))
  :state
  (r-a-s2 (rope-a-velocity downwards)))
```

Wheel *A* motion:

```
(def-quantity-behavior pulley
    (wheel-angular-motion wheel-velocity)
  :state
  (w-s1 (wheel-velocity 0))
  :transition
  (trans (w-s1 w-s2)
         :using-function
         ((allow wheel wheel-angular-motion))
         :under-condition-structure
```

```
        ((rolling-joint rope-a wheel))
        :under-condition-transition
        ((rope-a-motion r-a-s1 r-a-s2)))
  :state
  (w-s2 (wheel-velocity clockwise)))
```

Rope *B* motion:

```
(def-quantity-behavior pulley
    (rope-b-motion rope-b-velocity)
  :state
  (r-b-s1 (rope-b-velocity 0))
  :transition
  (trans (r-b-s1 r-b-s2)
        :using-function
        ((allow rope-b rope-b-motion))
        :under-condition-structure
        ((rolling-joint rope-b wheel))
        :under-condition-transition
        ((wheel-angular-motion w-s1 w-s2)))
  :state
  (r-b-s2 (rope-b-velocity upwards)))
```

And finally, the motion of the weight being lifted:

```
(def-quantity-behavior pulley
    (weight-motion weight-velocity)
  :state
  (m-s1 (weight-velocity 0))
  :transition
  (trans (m-s1 m-s2)
        :using-function
```

```
        ((allow weight weight-motion))
        :under-condition-structure
        ((attached rope-b weight))
        :under-condition-transition
        ((rope-b-motion r-b-s1 r-b-s2)))
  :state
  (m-s2 (weight-velocity upwards)))
```

Thus completing the behavioral model.

The functional specification, then, is:

```
(def-functional-spec pulley lift-weight
  :given (weight-motion m-s1)
  :makes (weight-motion m-s2)
  :by-behavior (behavior weight-motion))
```

Thus completing the DSSBF model for the simplified pulley example.

REFERENCES

[1] ALVARADO, C. and DAVIS, R., "Resolving ambiguities to create a natural computer-based sketching environment," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 1365–1371, Morgan Kaufmann Publishers, 2001.

[2] ALVARADO, C. and DAVIS, R., "SketchREAD: A multi-domain sketch recognition engine," in *Proceedings of the 17th ACM Symposium on User Interface Software and Technology (UIST-04)*, (New York, NY), pp. 23–32, ACM Press, 2004.

[3] ALVARADO, C. and DAVIS, R., "Dynamically constructed bayes nets for multi-domain sketch recognition," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 1407–1412, Morgan Kaufman Publishers, 2005.

[4] BARBER, J., JACOBSON, M., PENBERTHY, L., SIMPSON, R., BHATTA, S., GOEL, A., PEARCE, M., SHANKAR, M., and STROULIA, E., "Integrating artificial intelligence and multimedia technologies for interface design advising," *NCR Journal of Research and Development*, vol. 6, no. 1, pp. 75–85, 1992.

[5] BARKER-PLUMMER, D., COX, R., and SWOBODA, N., eds., *Diagrammatic Representation and Inference: 4th International Conference Proceedings (Diagrams 2006)*, no. 4045 in Lecture Notes in Artificial Intelligence, (Berlin), Springer, 2006.

[6] BHATTA, S. and GOEL, A. K., "A functional theory of design patterns," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 294–300, Morgan Kaufmann, 1997.

[7] BITNER, J. R. and REINGOLD, E. M., "Backtrack programming techniques," *Communications of the ACM*, vol. 18, no. 11, pp. 651–656, 1975.

[8] BLACKWELL, A., MARRIOTT, K., and SHIMOJIMA, A., eds., *Diagrammatic Representation and Inference: 3rd. International Conference, Diagrams 2004*, no. 2980 in Lecture Notes in Artificial Intelligence, (Berlin), Springer, 2004.

[9] BYLANDER, T., "A theory of consolidation for reasoning about devices," *International Journal of Man-Machine Studies*, vol. 35, no. 4, pp. 467–489, 1991.

[10] BYLANDER, T. and CHANDRASEKARAN, B., "Understanding behavior using consolidation," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 450–454, Morgan Kaufmann, 1985.

[11] CARBONELL, J. and VELOSO, M., "Integrating derivational analogy into a general problem solving architecture," in *Proceedings of the 1st Workshop on Case-Based Reasoning*, pp. 104–124, Morgan Kaufmann, 1988.

[12] CHANDRASEKARAN, B., GOEL, A. K., and IWASAKI, Y., "Functional representation as a basis for design rationale," *IEEE Computer*, vol. 26, no. 1, pp. 48–56, 1993.

[13] CHANDRASEKARAN, B., GOEL, A. K., and IWASAKI, Y., "Functional representation as design rationale," in *Proceedings of the European Workshop on Case-Based Reasoning*, (Berlin), pp. 58–75, Springer, 1993.

[14] CHEN, C.-W. K. and YUN, D. Y. Y., "Unifying graph matching problems with a practical solution," in *Proceedings of the International Conference on Systems, Signals, Control, and Computers*, 1998.

[15] CROFT, D. and THAGARD, P., "Dynamic imagery: A computational model of motion and visual analogy," in *Model-Based Reasoning: Science, Technology, and Values* (MAGNANI, L. and NERCESSIAN, N. J., eds.), pp. 259–274, New York: Kluwer Academic Publishers/Plenum Publishers, 2002.

[16] DAVIS, R., "Sketch understanding in design: Overview of work at the MIT AI lab," in *Sketch Understanding: Papers from the 2002 Spring Symposium* (DAVIS, R., LANDAY, J., and STAHOVICH, T. F., eds.), Technical Report SS-02-08, (Menlo Park, CA), pp. 24–31, AAAI Press, 2002.

[17] DAVIS, R., LANDAY, J., and STAHOVICH, T. F., eds., *Sketch Understanding: Papers from the 2002 Spring Symposium*, Technical Report SS-02-08, (Menlo Park, CA), AAAI Press, 2002.

[18] DE BERG, M., VAN KREVELD, M., OVERMARS, M., and SCHWARTZKOPF, O., *Computational Geometry: Algorithms and Applications*. Berlin: Springer, 2nd, revised ed., 2000.

[19] DE KLEER, J. and BROWN, J. S., "A qualitative physics based on confluences," *Artificial Intelligence*, vol. 24, pp. 7–83, 1984.

[20] DIESTEL, R., *Graph Theory*. No. 173 in Graduate Texts in Mathematics, Heidelberg: Springer, 3rd ed., 2005.

[21] DUNCKER, K., "A qualitative (experimental and theoretical) study of productive thinking (solving of comprehensive problems)," *Journal of Genetic Psychology*, 1926.

[22] EPPSTEIN, D., "Subgraph isomorphism in planar graphs and related problems," *Journal of Graph Algorithms and Applications*, vol. 3, no. 3, pp. 1–27, 1999.

[23] EVANS, T. G., "A heuristic program to solve geometric analogy problems," in *Semantic Information Processing* (MINSKY, M., ed.), Cambridge, MA: MIT Press, 1968.

[24] FALKENHAINER, B., "A unified approach to explanation and theory formation," in *Computational Models of Scientific Discovery and Theory Formation* (SHRAGER, J. and LANGLEY, P., eds.), ch. 6, pp. 157–196, Morgan Kaufmann, 1990.

[25] FALKENHAINER, B., FORBUS, K. D., and GENTNER, D., "The structure-mapping engine: Algorithm and examples," *Artificial Intelligence*, vol. 41, pp. 1–63, 1990.

[26] FALTINGS, B., "Qualitative kinematics in mechanisms," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)* (MCDERMOTT, J., ed.), pp. 436–442, Morgan Kaufmann Publishers, 1987.

[27] FERGUSON, R. W., "MAGI: Analogy-based encoding using regularity and symmetry," in *Proceedings of the 16th Annual Conference Cognitive Science Society*, (Mahwah, NJ), pp. 283–288, Lawrence Erlbaum Associates, 1994.

[28] FERGUSON, R. W. and FORBUS, K. D., "Telling juxtapositions: Using repetition and alignable difference in diagram understanding.," in *Advances in Analogy Research* (HOLYOAK, K., GENTNER, D., and KOKINOV, B., eds.), pp. 109–117, Sofia, Bulgaria: New Bulgarian University, 1998.

[29] FERGUSON, R. W. and FORBUS, K. D., "GeoRep: A flexible tool for spatial representation of line drawings," in *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, (Austin, Texas), AAAI Press, 2000.

[30] FORBUS, K. D., "Qualitative process theory," *Artificial Intelligence*, vol. 24, pp. 85–168, 1984.

[31] FORBUS, K. D., NIELSEN, P., and FALTINGS, B., "Qualitative kinematics: A framework," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 430–435, Morgan Kaufmann, 1987.

[32] FORBUS, K. D., NIELSEN, P., and FALTINGS, B., "Qualitative spatial reasoning: The CLOCK project," *Artificial Intelligence*, vol. 51, no. 1–3, pp. 417–471, 1991.

[33] FREUDER, E. C., "A sufficient condition for backtrack-free search," *Journal of the ACM*, vol. 29, no. 1, pp. 24–32, 1982.

[34] FREUDER, E. C., "A sufficient condition for backtrack-bounded search," *Journal of the ACM*, vol. 32, no. 4, pp. 755–761, 1985.

[35] FUNT, B. V., "Problem-solving with diagrammatic representations," *Artificial Intelligence*, vol. 13, no. 4, pp. 201–230, 1980.

[36] GEBHARDT, F., VOSS, A., GRÄTHER, W., and SCHMIDT-BELZ, B., *Reasoning with Complex Cases*, vol. 393 of *Kluwer Series in Engineering and Computer Science*. Boston: Kluwer Academic Publishers, 1997.

[37] GENTNER, D., "Structure-mapping: A theoretical framework for analogy," *Congitive Science*, vol. 7, no. 2, pp. 155–170, 1983.

[38] GERO, J. S. and MCNEILL, T., "An approach to the analysis of design protocols," *Design Studies*, vol. 19, no. 1, pp. 21–61, 1998.

[39] GERO, J. S., THAM, K. W., and LEE, H. S., "Behavior: A link between function and structure," in *Intelligent Computer Aided Design* (BROWN, D. C., WALDRON, M. B., and YOSHIKAWA, H., eds.), pp. 193–225, Amsterdam: North-Holland, 1992.

[40] GERO, J. S. and TVERSKY, B., eds., *Visual and Spatial Reasoning in Design*, Key Center of Design Computing and Cognition, University of Sydney, 1999.

[41] GERO, J. S., TVERSKY, B., and KNIGHT, T., eds., *Visual and Spatial Reasoning in Design III*, Key Center of Design Computing and Cognition, University of Sydney, 2004.

[42] GERO, J. S., TVERSKY, B., and PURCELL, T., eds., *Visual and Spatial Reasoning in Design II*, Key Center of Design Computing and Cognition, University of Sydney, 2001.

[43] GICK, M. L. and HOLYOAK, K. J., "Analogical problem solving," *Cognitive Psychology*, vol. 12, pp. 306–355, 1980.

[44] GLASGOW, J., NARAYANAN, N. H., and CHANDRASEKARAN, B., eds., *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. Menlo Park, CA: AAAI Press/The MIT Press, 1995.

[45] GLYSSENS, M., JEAVONS, P. G., and COHEN, D. A., "Decomposing constraint satisfaction problems using database techniques," *Artificial Intelligence*, vol. 66, no. 1, pp. 57–89, 1994.

[46] GOEL, A. K., BHATTA, S. R., and STROULIA, E., "Kritik: An early case-based design system," in *Issues and Applications of Case-Based Reasoning in Design* (MAHLER, M. and PU, P., eds.), pp. 87–132, Mahwah, New Jersey: Lawrence Erlbaum Associates, 1997.

[47] GOEL, A. K. and CHANDRASEKARAN, B., "Functional representation in design and redesign problem solving," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 1388–1394, Morgan Kaufmann, 1989.

[48] GRIFFITH, T. W., *A Computational Theory of Generative Modeling in Scientific Reasoning*. Phd dissertation, Georgia Institute of Technology, Atlanta, GA, 1999.

[49] HAMMOND, K. J., "CHEF: A model of case-based planning," in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, (Philadelphia, PA), AAAI Press, 1986.

[50] HEGARTY, M., "Mental animation: Inferring motion from static displays of mechanical systems," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 18, no. 5, pp. 1084–1102, 1992.

[51] HEGARTY, M., MEYER, B., and NARAYANAN, N. H., eds., *Diagrammatic Representation and Inference: 2nd International Conference, Diagrams 2002*, no. 2317 in Lecture Notes in Artificial Intelligence, (Berlin), Springer, 2002.

[52] HOFSTADTER, D. and MCGRAW, G., "Letter spirit: Esthetic perception and creative play in the rich microcosm of the roman alphabet," in *Fluid Concepts and Creative Analogies: Computer Models of the Funamental Mechanisms of Thought* (HOFSTADTER, D. and THE FLUID ANALOGIES RESEARCH GROUP, eds.), ch. 10, pp. 407–466, Basic Books, 1995.

[53] HOLYOAK, K. J. and THAGARD, P., "Analogical mapping by constraint satisfaction," *Cognitive Science*, vol. 13, no. 3, pp. 295–355, 1989.

[54] HOLYOAK, K. J. and THAGARD, P., "A computational model of analogical problem solving," in *Similarity and Analogical Reasoning* (VOSNIADOU, S. and ORTONY, A., eds.), ch. 8, pp. 242–266, Cambridge University Press, 1989.

[55] KONDRAK, G. and VAN BEEK, P., "A theoretical evaluation of selected backtracking algorithms," *Artificial Intelligence*, vol. 89, no. 1-2, pp. 365–387, 1997.

[56] KUIPERS, B., "Commonsense reasoning about causality: Deriving behavior from structure," *Artificial Intelligence*, vol. 24, pp. 169–203, 1984.

[57] KUIPERS, B., *Qualitative Reasoning: Modeling and Simulation With Incomplete Knowledge*. Cambridge, MA: MIT Press, 1994.

[58] KUMAR, V., "Algorithms for constraint satisfaction problems: A survey," *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992.

[59] KURTOGLU, T. and STAHOVICH, T. F., "Interpreting schematic sketches using physical reasoning," in *Sketch Understanding: Papers from the 2002 Spring Symposium* (DAVIS, R., LANDAY, J., and STAHOVICH, T. F., eds.), Technical Report SS-02-08, (Menlo Park, CA), pp. 78–85, AAAI Press, 2002.

[60] LARKIN, J. H. and SIMON, H. A., "Why a diagram is (sometimes) worth ten thousand words," *Cognitive Science*, vol. 11, pp. 65–99, 1987.

[61] MCGRAW, G. and HOFSTADTER, D., "Perception and creation of diverse alphabetic styles," in *Artificial Intelligence and Creativity: Papers from the 1993 Spring Symposium* (HARRISON, S., ed.), Technical Report SS-93-01, pp. 11–18, Menlo Park, California: AAAI Press, 1993.

[62] MEDIN, D. L., GOLDSTONE, R. L., and GENTNER, D., "Respects for similarity," *Psychological Review*, vol. 100, no. 2, pp. 254–278, 1993.

[63] NIELSEN, P., "A qualitative approach to mechanical constraint," in *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pp. 270–274, AAAI Press, 1988.

[64] O'HARA, S. and INDURKHYA, B., "Incorporating (re)-interpretation in case-based reasoning," in *Topics in Case-Based Reasoning: First European Workshop (EWCBR-93)* (WESS, S., ALTHOFF, K.-D., and RICHTER, M. M., eds.), vol. 837 of *Lecture Notes in Computer Science*, pp. 246–260, Springer, 1994.

[65] PEARCE, M., GOEL, A. K., KOLODNER, J. L., ZIMRING, C., SENTOSA, L., and BILLINGTON, R., "Case-based design support: A case study in architectural design," *IEEE Expert: Intelligent Systems & their Applications*, vol. 7, no. 5, pp. 14–20, 1992.

[66] RASMUSSEN, J., "The role of hierarchical knowledge representation in decision making and system management," *IEEE Trans. Systems, Man and Cybernetics*, vol. 15, pp. 234–243, 1985.

[67] SCHWARTZ, D. L. and HEGARTY, M., "Coordinating multiple representations for reasoning about mechanical devices," in *Cognitive and Computational Models of Spatial Representation: Papers from the 1996 Spring Symposium* (OLIVIER, P., ed.), Technical Report SS-96-03, (Menlo Park, CA), pp. 101–109, AAAI Press, 1996.

[68] SEMBUGAMOORTHY, V. and CHANDRASEKARAN, B., "Functional representation of devices and compilation of diagnostic problem-solving systems," in *Experience, Memory, and Reasoning* (KOLODNER, J. and RIESBECK, C., eds.), pp. 47–73, Hillsdale, NJ: Lawrence Erlbaum, 1986.

[69] SEZGIN, T. M. and DAVIS, R., "Sketch interpretation using multiscale models of temporal patterns," *IEEE Computer Graphics and Applications*, vol. 27, no. 1, pp. 28–37, 2007.

[70] SLOMAN, A., "Musings on the roles of logical and nonlogical representations in intelligence," in *Diagrammatic Reasoning: Cognitive and Computational Perspectives* (GLASGOW, J., NARAYANAN, N. H., and CHANDRASEKARAN, B., eds.), pp. 7–32, Menlo Park, CA: AAAI Press/The MIT Press, 1995.

[71] STAHOVICH, T. F., DAVIS, R., and SHROBE, H., "Generating multiple new designs from a sketch," *Artificial Intelligence*, vol. 104, no. 1–2, pp. 211–264, 2001.

[72] STINY, G., "Introduction to shape and shape grammars," *Environment and Planning B: Planning and Design*, vol. 7, no. 3, pp. 343–351, 1980.

[73] STINY, G., "Shape rules: Closure, continuity, and emergence," *Environment and Planning B: Planning and Design*, vol. 21, no. 7, pp. s49–s78, 1994.

[74] STINY, G., *Shape: Talking About Seeing and Doing*. Cambridge, MA: MIT Press, 2006.

[75] STINY, G. and GIPS, J., "Shape grammars and the generative specification of painting and sculpture," in *Proceedings of IFIP Congress 71* (FREIMAN, C. V., ed.), (Amsterdam), pp. 1460–1465, North-Holland, 1972.

[76] SUETENS, P., FUA, P., and HANSON, A. J., "Computational strategies for object recognition," *ACM Computing Surveys*, vol. 24, no. 1, pp. 5–61, 1992.

[77] TAPIA, M., "A visual implementation of a shape grammar system," *Environment and Planning B: Planning and Design*, vol. 26, no. 1, pp. 59–73, 1999.

[78] THAGARD, P., GOCHFELD, D., and HARDY, S., "Visual analogical mapping," in *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, pp. 522–527, Lawrence Erlbaum Associates, 1992.

[79] THAGARD, P., HOLYOAK, K. J., NELSON, G., and GOCHFELD, D., "Analog retrieval by constrain satisfaction," *Artificial Intelligence*, vol. 46, pp. 259–310, 1990.

[80] ULLMAN, J. R., "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[81] UMEDA, Y., TAKEDA, H., TOMIYAMA, T., and YOSHIKAWA, H., "Function, behavior, and structure," in *Proceedings of the 5th International Conference on Applications of AI in Engineering*, vol. 1, (Boston, MA), pp. 177–193, Springer, 1990.

[82] VELOSO, M. and CARBONELL, J., "Learning analogies by analogy - the closed loop of memory organization and problem solving," in *Proceedings of the 2nd Workshop on Case-Based Reasoning*, pp. 153–158, Morgan Kaufmann, 1989.

[83] VELOSO, M., CARBONELL, J., ALICIA, P., BORRAJO, D., FINK, E., and BLYTHE, J., "Integrating planning and learning: The PRODIGY architecture," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 7, no. 1, 1995.

[84] WINSTON, P. H., *Artificial Intelligence*. Reading, MA: Addison-Wesley, third ed., 1992.

[85] YANER, P. W. and GOEL, A. K., "From diagrams to models by analogical transfer," in *Diagrammatic Representation and Inference: 4th International Conference Proceedings (Diagrams 2006)* (BARKER-PLUMMER, D., COX, R., and SWOBODA, N., eds.), no. 4045 in Lecture Notes in Artificial Intelligence, (Berlin), pp. 55–69, Springer, 2006.

[86] YANER, P. W. and GOEL, A. K., "Visual analogy: Viewing analogical retrieval and mapping as constraint satisfaction problems," *Applied Intelligence*, vol. 25, no. 1, pp. 91–105, 2006.

[87] YANER, P. W. and GOEL, A. K., "Understanding drawings by compositional analogy," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 1131–1137, Morgan Kaufmann, 2007.